



Architecture des ordinateurs et assembleur

2022/2023

I. Représentation de l'information

II. Architecture de base d'un ordinateur

III. Algèbre de Boole & Logiques Combinatoire et Séquentielle

IV. Structure interne des Microprocesseurs

V. Programmation en assembleur

Introduction

Qu'est ce qu'un microprocesseur ?

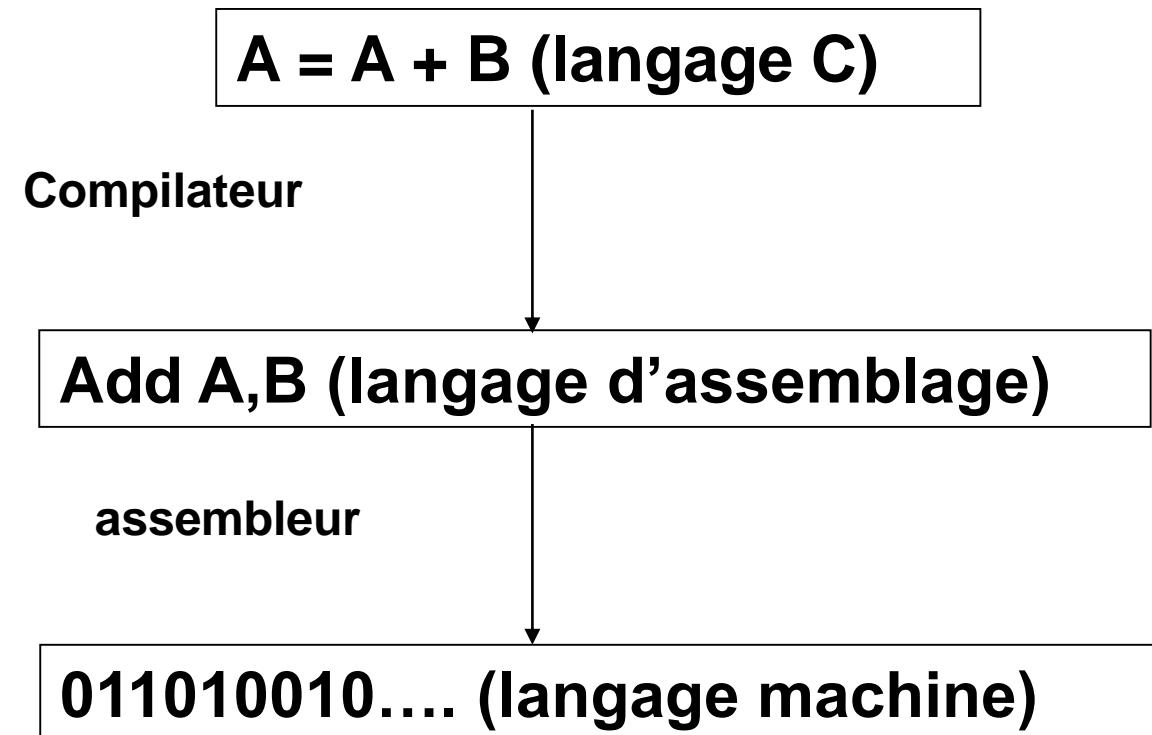
- Un microprocesseur (**μP**) est un circuit intégré complexe.
 - Micro=petite taille
 - Processeur = calculateur
- Il résulte de l'intégration sur une puce de fonctions logiques
 - combinatoires (logiques et/ou arithmétique)
 - Séquentielles (registres, compteur, etc....).
- Son domaine d'utilisation est donc presque illimité.

Rôle du microprocesseur

- Le microprocesseur est capable d'interpréter et d'exécuter les instructions d'un **programme**.
- Un **programme** est une suite **d'instructions** qui réalise une tâche
- Une **instruction** est une opération simple
 - Opération de lecture ou écriture en mémoire
 - Opération logique (ET, OU, décalage ...)
 - Opération arithmétique (Addition, soustraction ...)
- Il existe plusieurs langages de programmation :
 - Assembleur (langage machine)
 - Langage évolué (C , Java ...)

Compilateur / Assembleur

La programmation de la machine se fait généralement en utilisant un langage de haut niveau (C, java,...)



Caractéristiques du µP

Le format des données

= Nombre de bit du bus de donnée

- 8bits
- 16bits
- 32bits
- 64bits

La puissance de traitement

S'exprime en **MIPS**
(Millions d'Instructions Par Seconde)

La puissance consommée

La taille de l 'espace adressable

= Nombre de bit du bus d 'adresse

- **16 bits = 65.536 adresses**
- **32 bits = 4.294.967.296 adresses**
- **64 bits = 18.446.744.073.709.551.616 adresses**

Le jeu d 'instructions

- **Etendu (CISC)** (Complex Instruction Set Computer)
- **Réduit (RISC)** (Reduced Instruction Set Computer)

Les performances

- La performance d'un microprocesseur pour un utilisateur individuel correspond à la durée s'écoulant entre le début et la terminaison d'une tâche: *temps d'exécution*

$$\text{TempsD'exécution} = \text{NombreDeCycles} \times \text{TempsDeCycle}$$

Cycle Par Instruction
$$CPI = \frac{\text{NombreDeCyclesUCPourUnProgramme}}{NI}$$

$$\text{TempsUC} = NI \times CPI \times \text{TempsDeCycle}$$

- la **fréquence** de fonctionnement ne suffit pas pour comparer les performances
- le nombre de cycles par instruction dépend de la complexité moyenne du jeu d'instructions
- L'ensemble des améliorations des **microprocesseurs** visent à **diminuer** le temps d'exécution du programme.

Fonctions essentielles d'un microprocesseur

Ce circuit remplit deux fonctions essentielles :

- le traitement des données

On parle **d'unité de traitement**. Cette fonction est dédiée à l'U.A.L.
Elle concerne la **manipulation** des données sous formes de transfert, opérations arithmétiques, opérations logiques....

- le contrôle du système

Cette fonction se traduit par des opérations de **décodage** et **d'exécution** des ordres exprimés sous forme des instructions.

Puissance d'un microprocesseur

- La notion de puissance est la capacité de traiter un grand nombre d'opérations par seconde et en grande quantité.
- La puissance dépend des trois critères suivants:
 - **La longueur des mots** : données et instructions (on parle de largeur du bus des données).
 - **Le nombre d'octets** que le microprocesseur peut adresser (on parle de largeur du bus des adresses).
 - **La vitesse d'exécution des instructions** liée à la fréquence de fonctionnement de l'horloge de synchronisation exprimée en MHZ.

Système à base du microprocesseur

On distingue 3 éléments logiques principaux :

- **Une Unité Arithmétique et Logique (U.A.L.)**
- **Un Accumulateur.**
- **Des registres** que l'on nomme couramment :
 - Le Compteur d'Instructions (C.I.)
 - Le Registre d'état
 - Le Registre d'Instructions (R.I.)
 - Le Registre d'Adresses (R.A.)
 - Le Registre temporaire des données
- De base, il existe 6 registres fondamentaux dans une architecture de microprocesseur 8 bits.

Système à base du microprocesseur

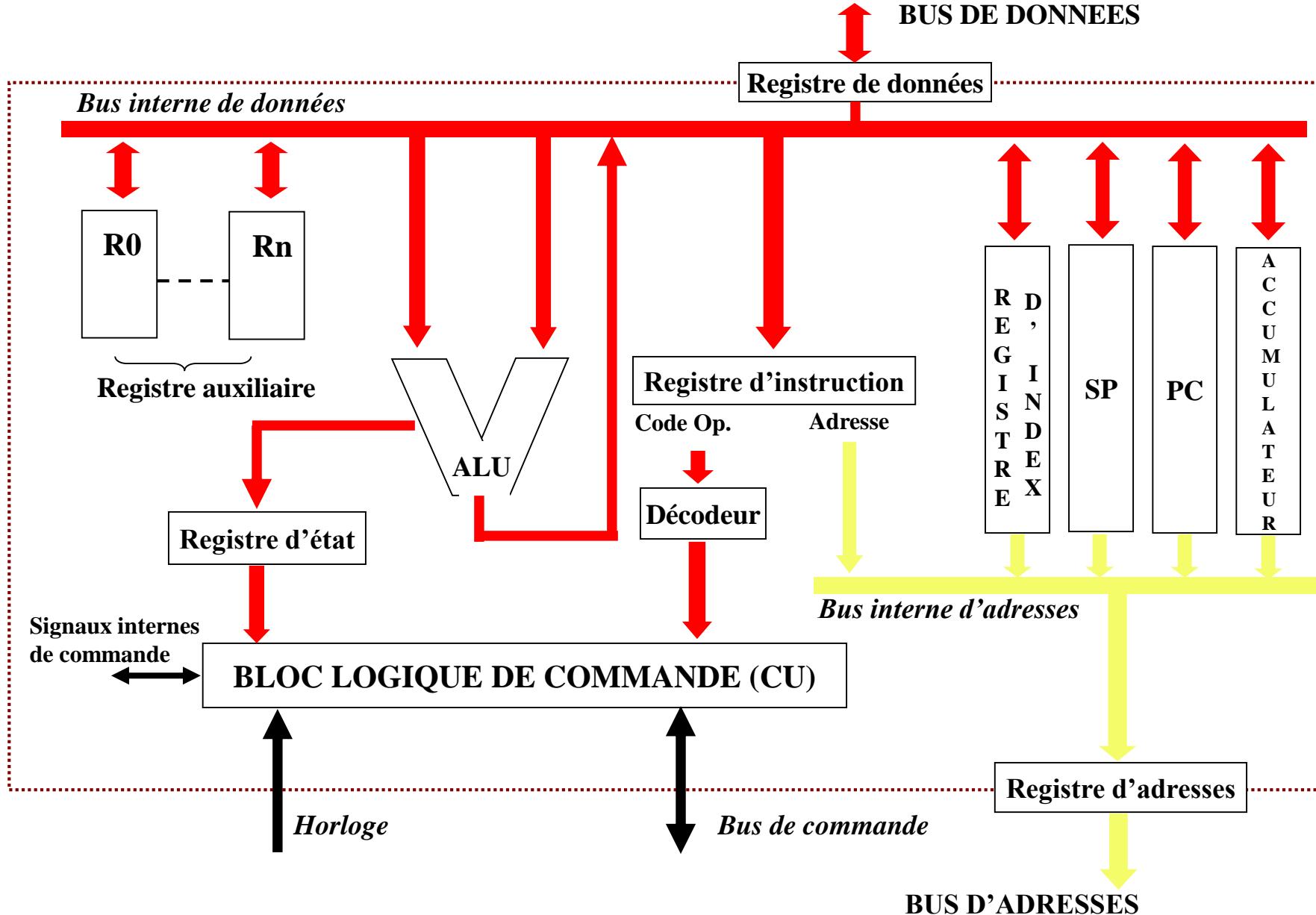
- Des registres supplémentaires sont ajoutés pour rendre la vie plus facile aux programmeurs
- Cet ensemble est interconnecté au travers de différents bus.
- On trouve trois types de bus :
 - Le bus des données (bi-directionnel)
 - Le bus des adresses (uni-directionnel)
 - Le bus de contrôle (bi-directionnel)

Rmq: le bus interne de données relie tous les différents éléments du micro-processeur.

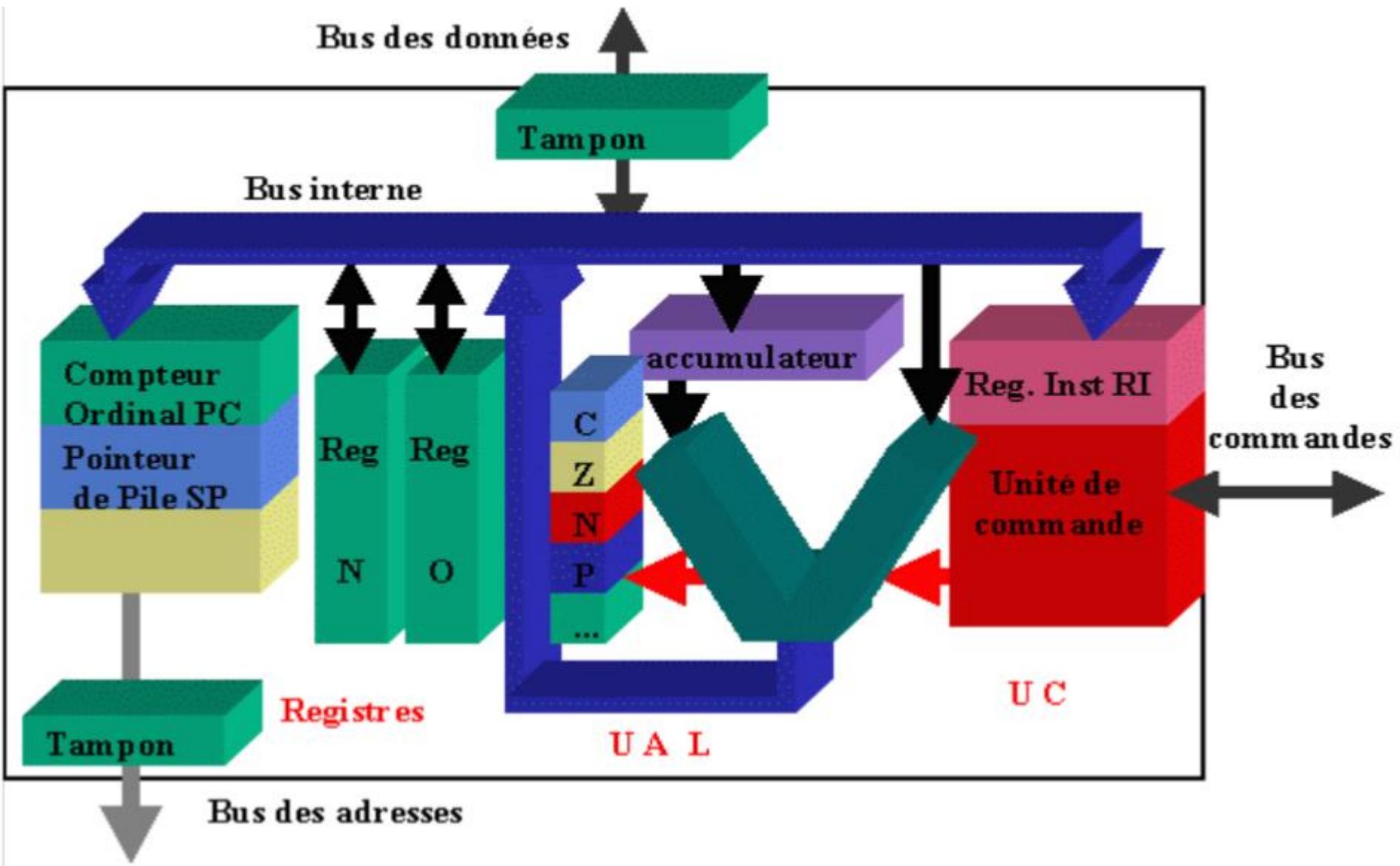
Le microprocesseur

- Lorsque tous les éléments suivants sont regroupés sur une même puce, on parle alors de **microprocesseur**:
 - une **unité de commande** qui lit les instructions et les décode;
 - une **unité de traitement** (UAL - unité arithmétique et logique) qui exécute les instructions;
 - d'un **ensemble de mémoire** appelés **registres**;
 - d'un **bus de données externe**;
 - d'un **bus d'adresse externe**;
 - d'un **bus de commande externe**;
 - d'un **bus de données interne** reliant l'unité de commande, l'UAL et les registres.

Architecture de base de microprocesseur



Architecture de base de microprocesseur



Registre de données

- ▶ Ce registre est un registre tampon qui assure l'interfaçage entre le microprocesseur et son environnement ou inversement.
- ▶ Il conditionne le bus externe ou le bus interne des données.

Registre d'adresses

- ▶ **Ce registre est un registre tampon qui assure l'interfaçage entre le microprocesseur et son environnement.**

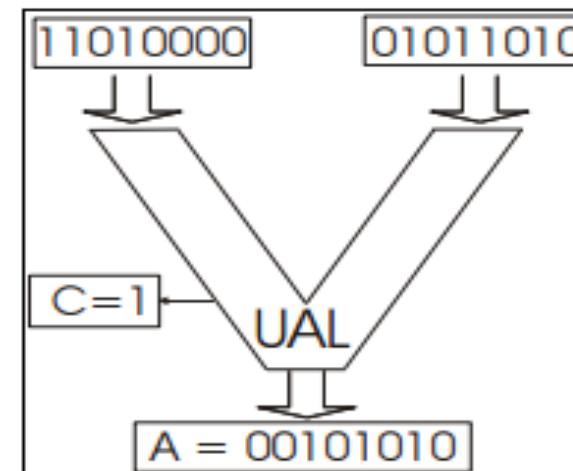
- ▶ **Il conditionne le bus externe des adresses.**

L'unité de commande (**Bloc logique de commande**)

- Elle permet de "séquencer" le déroulement des instructions.
- Il organise l'exécution des instructions au rythme d'une horloge.
- Elle élabore tous les signaux de synchronisation internes ou externes (bus des commandes) au microprocesseur.

L'unité arithmétique et logique (UAL)

- C'est un circuit complexe qui assure les fonctions:
 - arithmétiques: addition et soustraction
 - logiques: ET, OU, OU exclusif
 - comparaison, décalage à droite ou à gauche, incrémentation,
- Deux registres sont associés à l'UAL :
 - l'accumulateur
 - le registre d'état.



L'accumulateur (nommé : A)

- Un accumulateur est un registre de travail de 8 ou 16 bits
- C'est une des deux entrées de l'UAL. Il est impliqué dans presque toutes les opérations réalisées par l'UAL.
- Certains constructeurs ont des microprocesseurs à deux accumulateurs (Motorola : 6800).
- Exemple: A étant l'accumulateur et B un registre, on peut avoir : $A+B$ (ADD A,B : addition du contenu du registre A avec celui du registre B, le résultat étant mis dans A)

Le registre d'état (Flags : F)

- A chaque opération, le microprocesseur positionne un certain nombre de bascules d'état.
- Ces bascules sont appelées aussi indicateurs d'état ou drapeaux (status, flags).
- Par exemple, si une soustraction donne un résultat nul, l'indicateur de zéro (Z) sera mis à 1.
- Ces bascules sont regroupées dans le registre d'état

Le registre d'état (Flags : F)

- On peut citer comme indicateurs:
 - retenue (carry : C)
 - retenue intermédiaire (Auxiliary-Carry : AC)
 - signe (Sign : S)
 - débordement (Overflow : OV ou V)
 - Zéro (Z)
 - parité (Parity : P)

Retenue : (carry : C)

- Ce bit est dans l'état actif lorsque le huitième bit du résultat de l'opération génère une retenue.

Retenue : (carry : C)

Exemple: addition de nombres binaire sur 8 bits

$$\begin{array}{r} 11111100 \\ + \quad 10000010 \\ \hline \text{carry : } 1 = \quad 01111110 \end{array} \qquad \begin{array}{r} FCH \\ + \quad 82H \\ \hline \text{carry : } = \quad 7EH \end{array}$$

- **Exemple 2:**

soit le décalage à gauche d'un bit de cet octet : 10010110, après le décalage nous avons 00101100. En ce qui concerne le carry, il va être positionné à 1, puisque le bit expulsé est égal à 1.

Retenue intermédiaire : (Auxiliary Carry : AC)

- Sur les opérations arithmétiques, ce bit signale une retenue entre groupes de 4 bits (Half-byte: demi-octet) d'une quantité de 8 bits.

- **Exemple:**

- $\begin{array}{r} 0000\textcolor{red}{1}1010 \\ + 0010 \ 1010 \\ \hline \end{array}$

0011 0100

Signe: (S)

- Ce bit est mise à 1 lorsque le résultat de l'opération est négatif (MSB: bit de plus fort poids du résultat: à 1).

Débordement : (overflow : OV)

- Cet indicateur est mis à 1, lorsqu'il y a un dépassement de capacité pour les opérations arithmétiques en complément à 2.
- Sur 8 bits, on peut coder de -128 (1000 0000) à +127 (0111 1111).

Le bit Zéro : (Zéro : Z)

- Ce bit est actif lorsque l'opération a pour effet de mettre tous les bits d'un accumulateur ou d'un registre à la valeur logique 0 (très utilisé pour réaliser des compteurs).

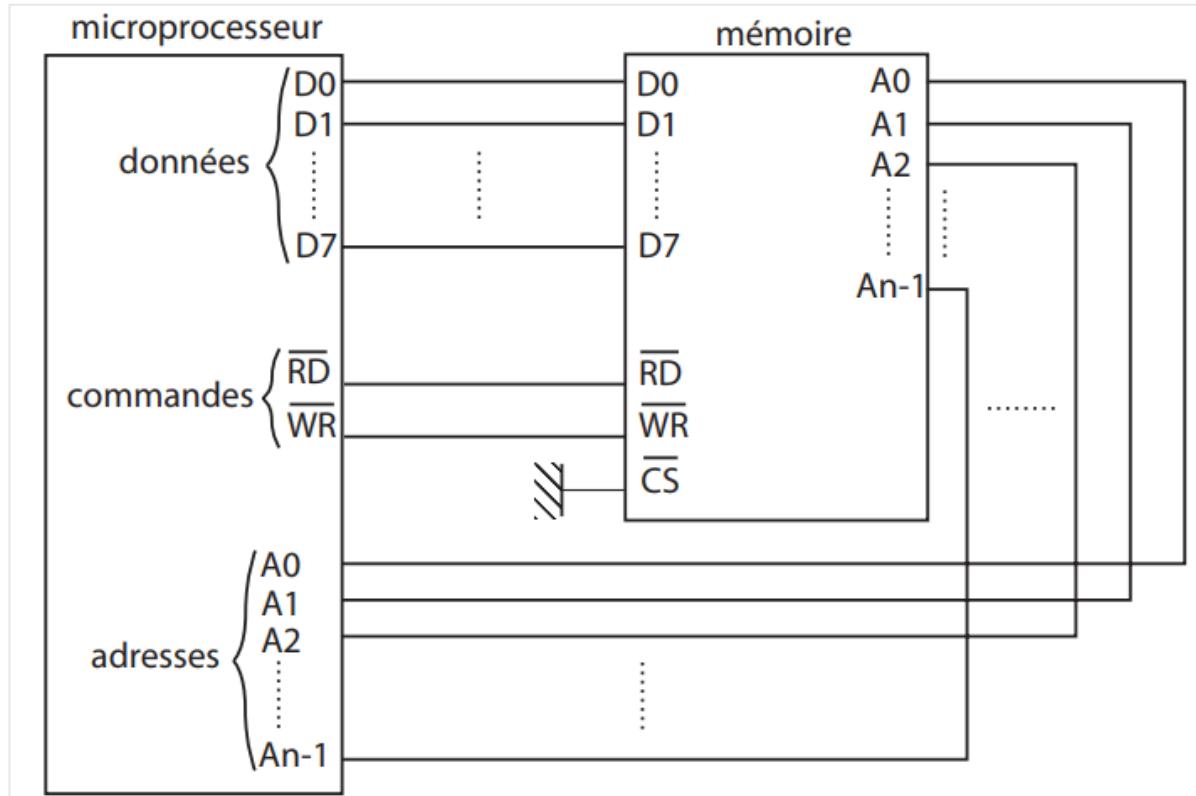
Le bit de parité : (P)

- Ce bit est mis à 1 lorsque le nombre de 1 de l'accumulateur est pair.

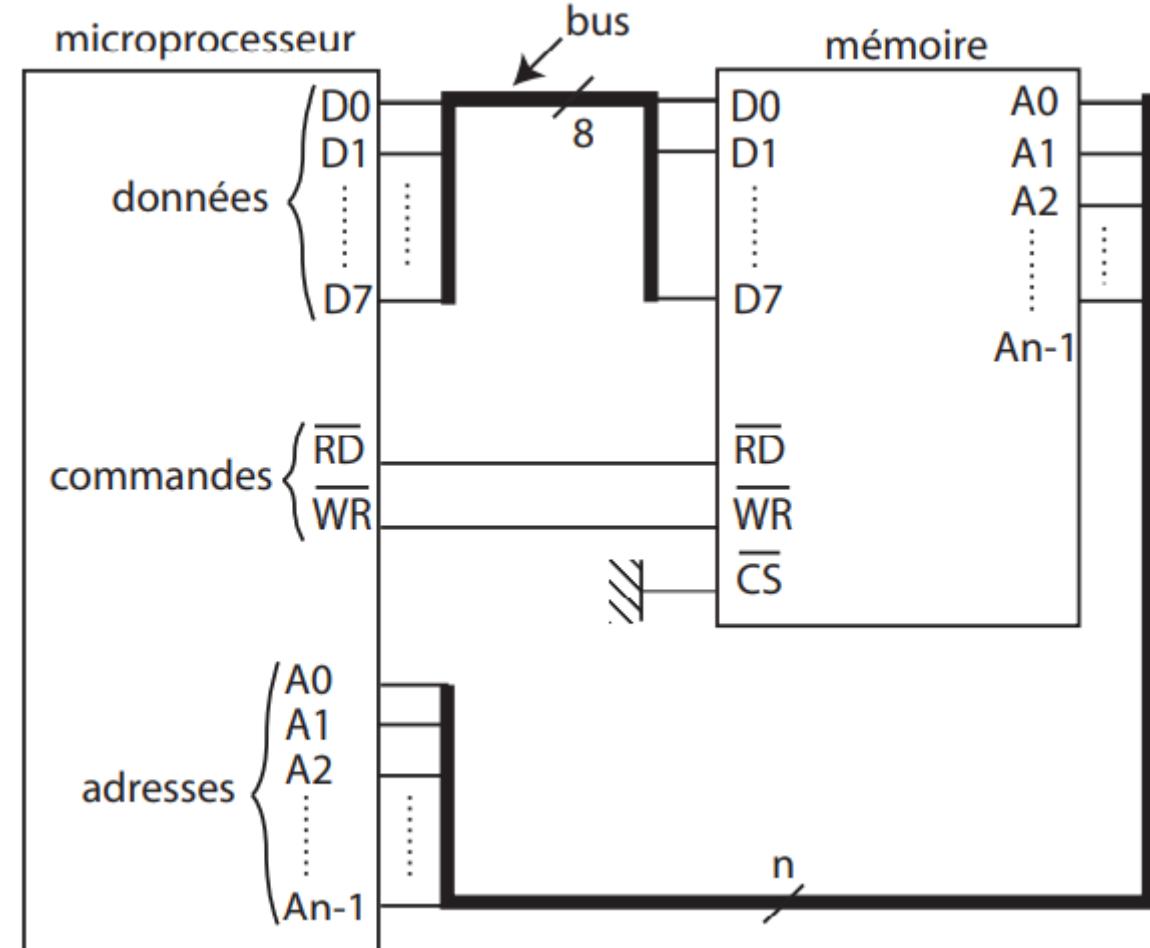
Remarque

- La plupart des instructions modifient **le registre d'état**.
- Exemple :
- **ADD A, B** positionne les drapeaux : O, S, Z, P, C
- **OR B, C** (B ou C -> B) positionne S, Z, P
- **MOV A, B** (Move, Transférer le contenu de B dans A) n'en positionne aucun.

Interfaçage microprocesseur/mémoire

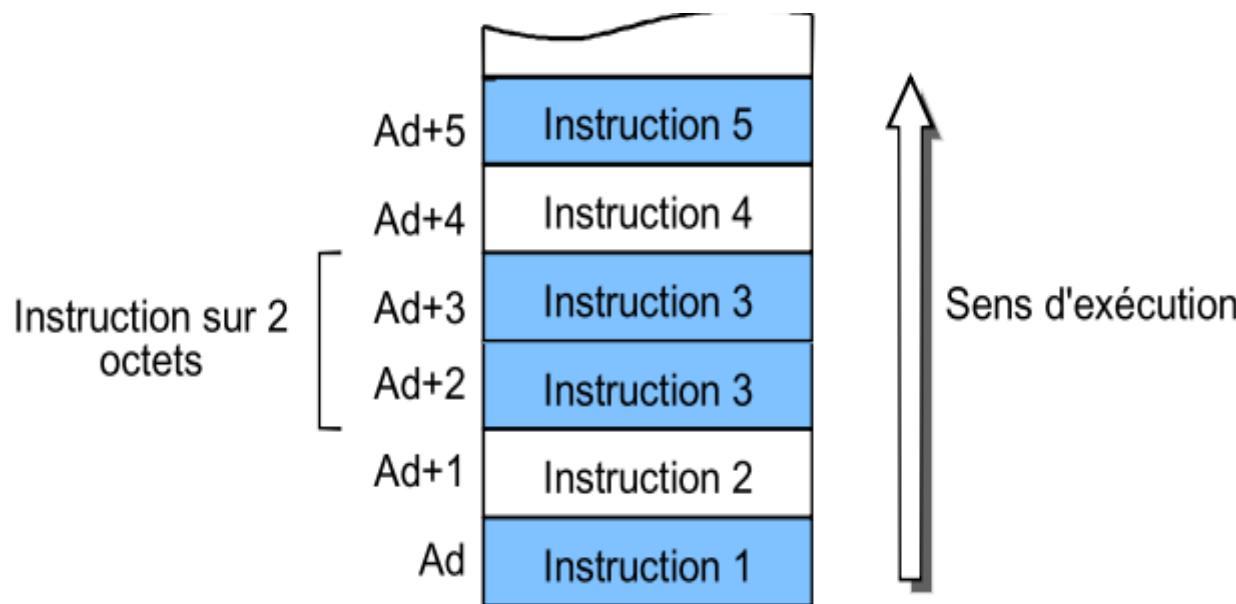


Représentation condensée (plus pratique)

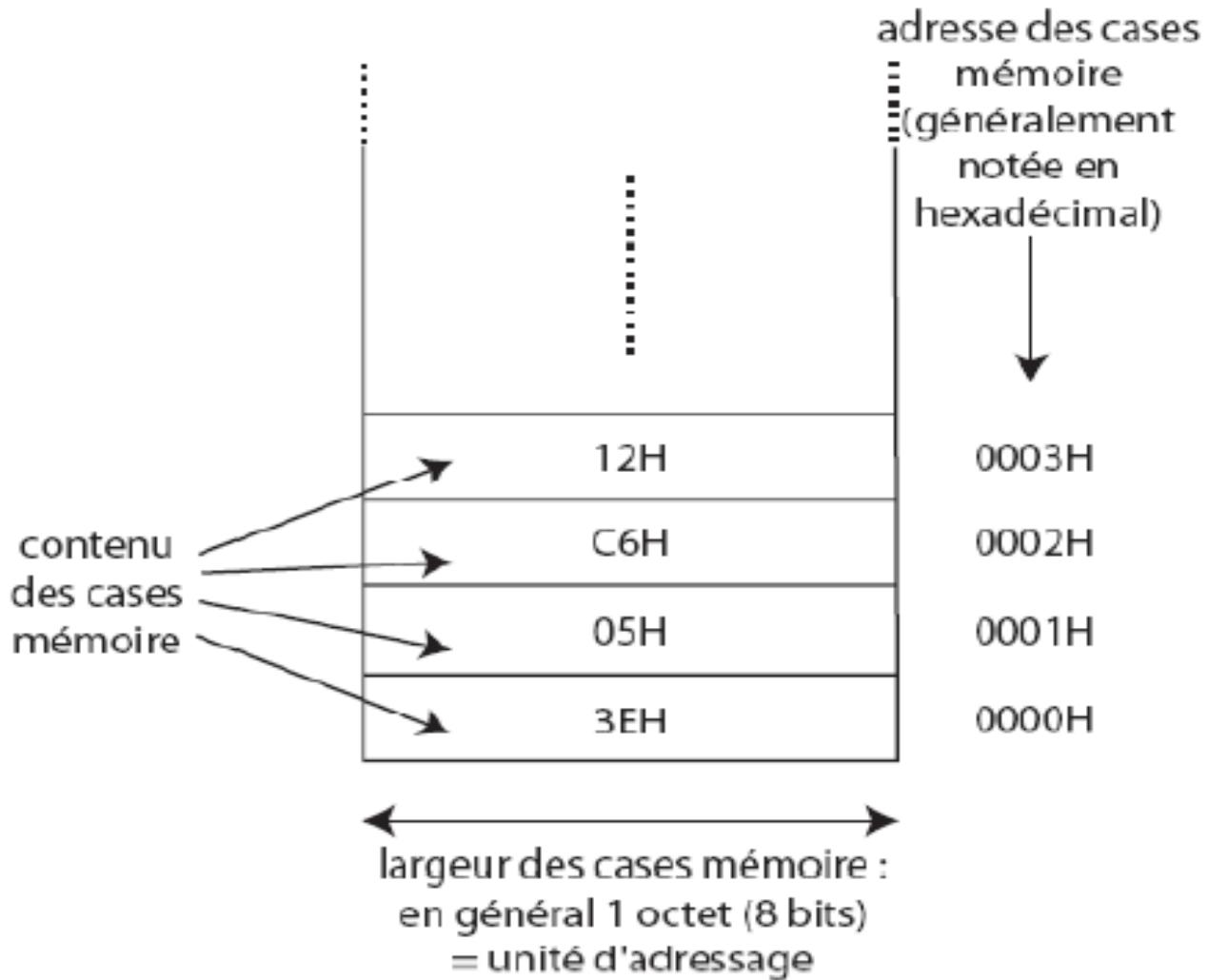


Rangement des instructions dans la mémoire programme

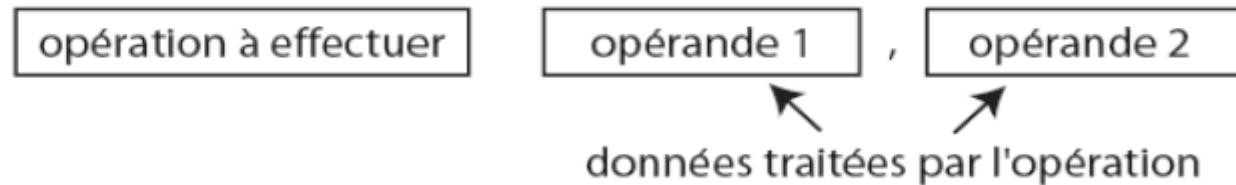
- Le µP est donc obligatoirement associé à une mémoire (mémoire programme) qui contient les codes des instructions à exécuter rangés dans des mots mémoires à des adresses successives (sauf en cas de rupture de séquence).



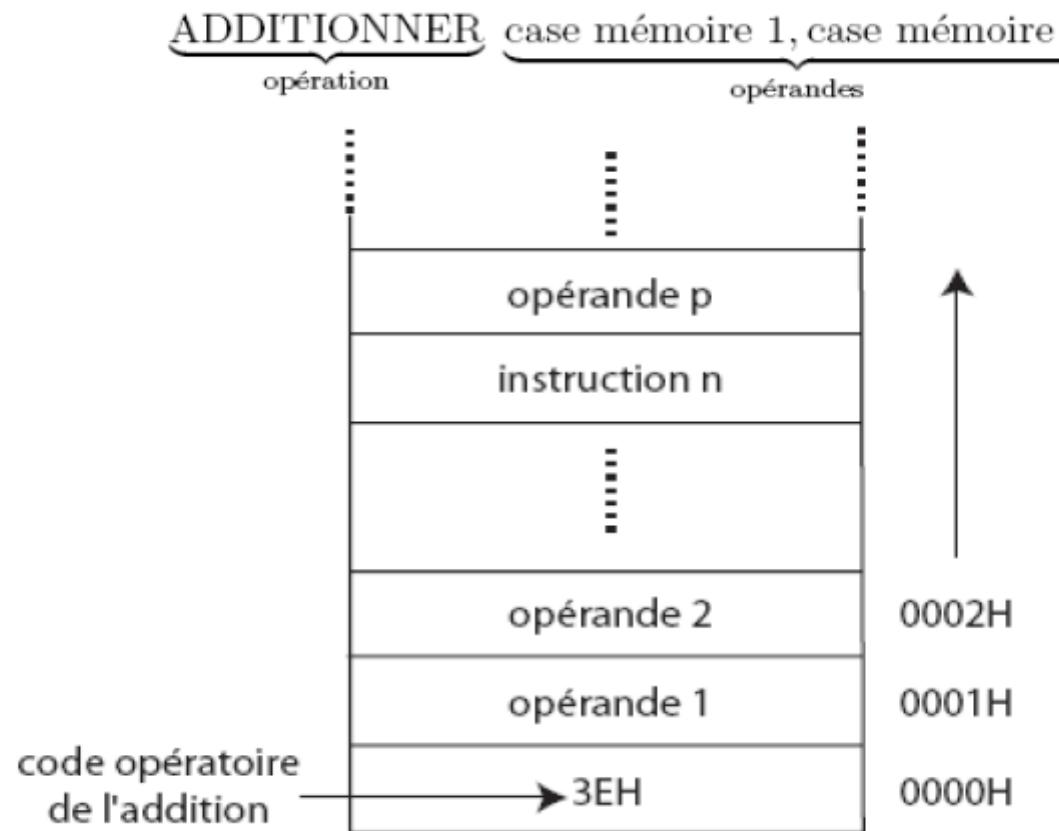
Organisation de la mémoire centrale



Format et arrangement d'une instruction



Exemple :



Les registres

- Il y'a deux type de registres :
 - les registres d'usage général (**Registres auxiliaires**)
 - les registres d'adresses (pointeurs).

Les registres d'usage général

- Ce sont des mémoires rapides, à l'intérieur du microprocesseur, qui permettent à l'UAL de manipuler des données à vitesse élevée. Ils sont connectés au bus de données interne du microprocesseur.
- L'adresse d'un registre est associée à son nom (on donne généralement comme nom une lettre) A, B,C...
- Exemple :
 - **MOV C,B** : transfert du contenu du registre B dans le registre C.

Les registres d'adresses (pointeurs)

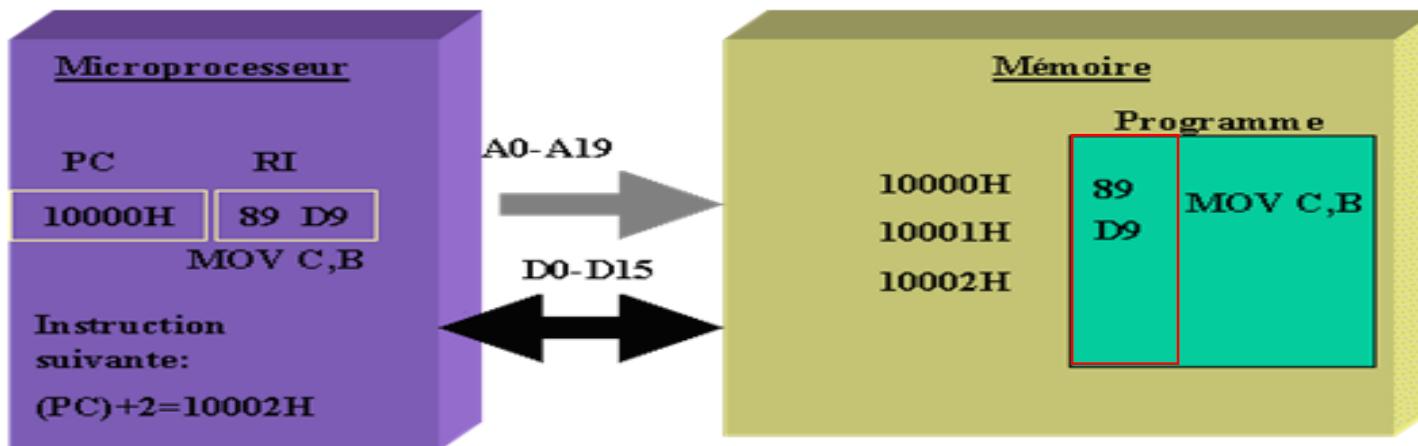
- Ce sont des registres connectés sur le bus d'adresses. On peut citer comme registre:
 - Le compteur ordinal (pointeur de programme PC) ;
 - Le pointeur de pile (stack pointer SP) ;
 - Les registres d'index (index source SI et index destination DI).

Le compteur ordinal (pointeur de programme PC)

- Il contient l'adresse de l'instruction à rechercher en mémoire. L'unité de commande incrémente le compteur ordinal (*program Counter :PC*) du nombre d'octets sur lequel l'instruction, en cours d'exécution, est codée.
- Le compteur ordinal contiendra alors l'adresse de l'instruction suivante.

Compteur de Programme (PC)

- Exemple : $(PC)=10000H$; il pointe la mémoire qui contient l'instruction `MOV CX,BX` qui est codée sur deux octets (89 D9H) ; l'unité de commande incrémentera de deux le contenu du PC : $(PC) = 10002H$



PC:

RI:

CX et BX:

A0-A19:

D0-D15:

Compteur de programme

Registre d'instruction

Registres d'usage général

lignes d'adresse sur 20 Bits

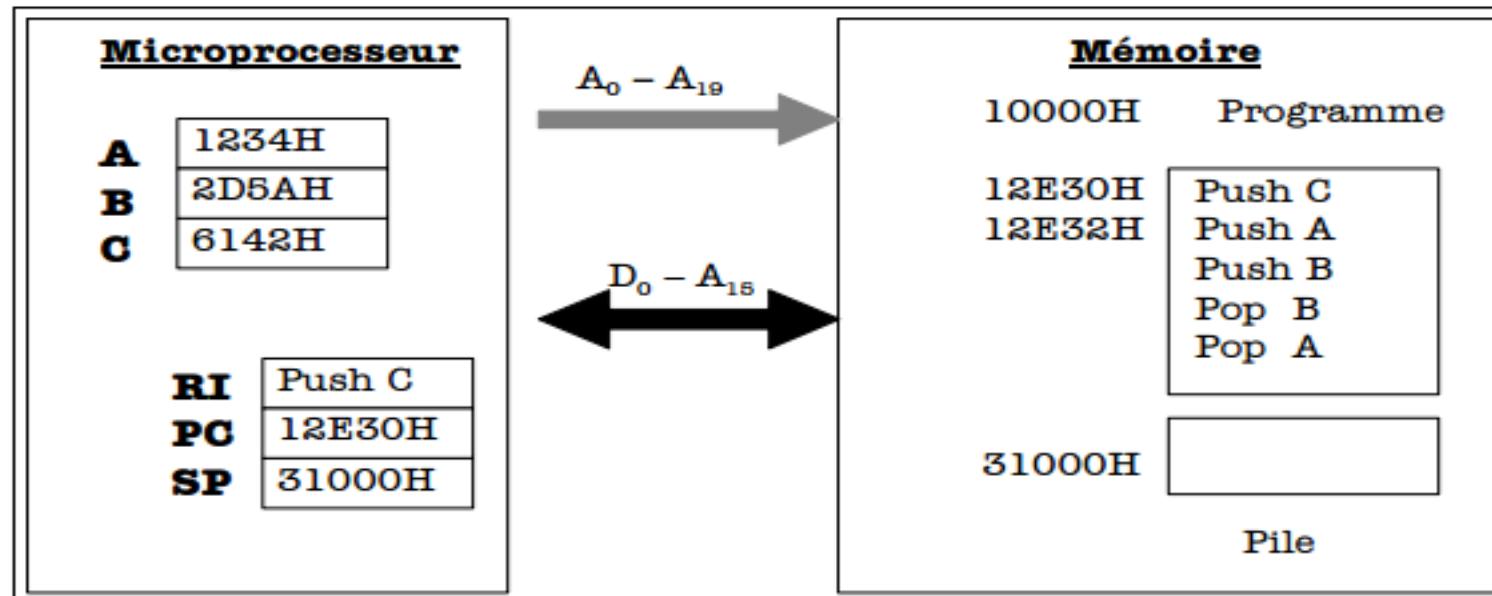
lignes de données sur 16 Bits

Le pointeur de pile (stack pointer SP)

- Il contient l'adresse de la pile. Celle-ci est une partie de la mémoire qui permet de stocker des informations (le contenu des registres) relatives au traitement des interruptions et des sous programmes.
- La pile est gérée en LIFO : (Last IN First Out)
- Le pointeur de pile SP pointe le haut de la pile (il est décrémenté avant chaque empilement, et incrémenté après chaque dépilement).
- Il existe deux instructions pour empiler et dépiler: PUSH et POP.
 - exemple: **PUSH A** empilera le registre A et **POP A** le dépilera

Exercice

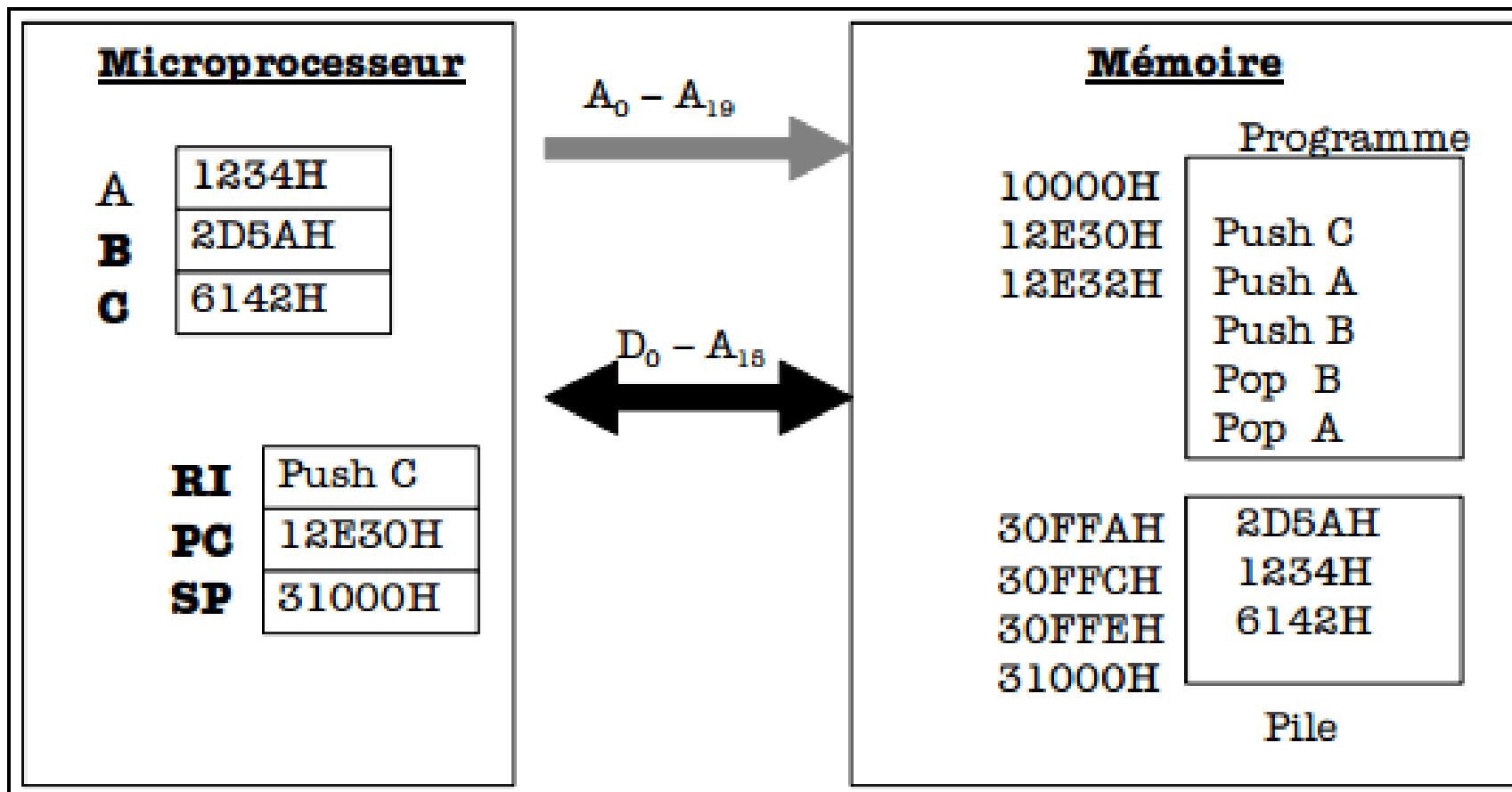
- Que se passera-t-il durant l'exécution du programme commençant en 12E30H? Que vaudra SP et que contiendra la pile à cette adresse à la fin du programme?



Réponse

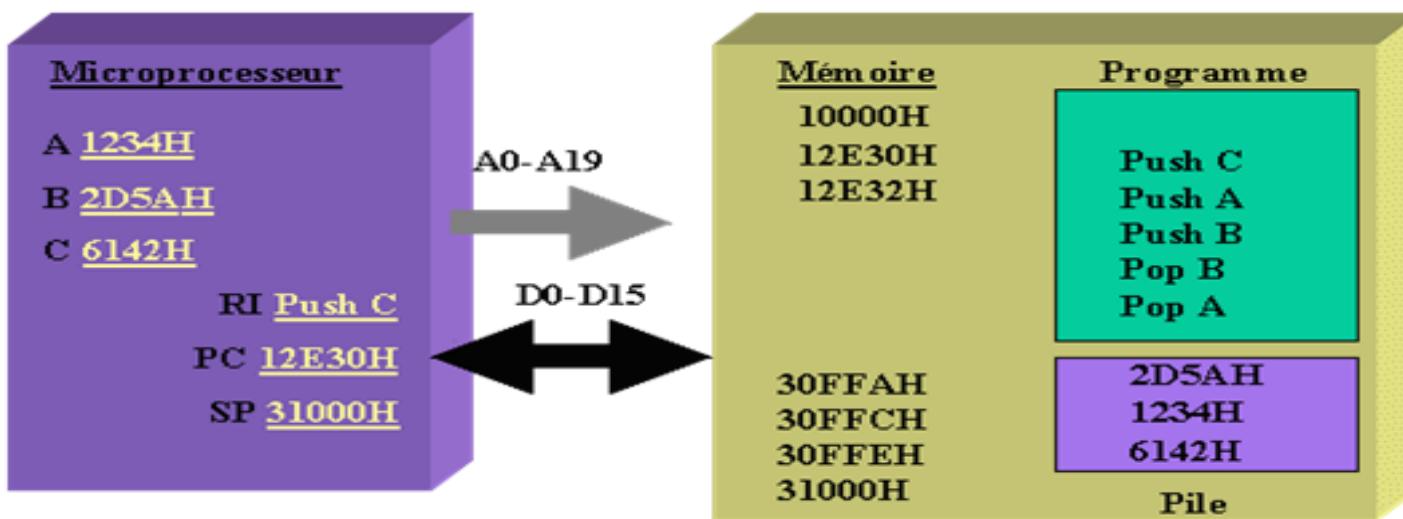
- Le programme commence par sauvegarder le contenu de registre C dans la pile (PUSH C).
- Pour cela (SP) est décrémenté de deux ($(SP)=31000H - 2 = 30FFEH$), ensuite le contenu du registre (C) sera chargé dans la mémoire à l'adresse 30FFEH.
- Pour PUSH A on obtient : $(30FFCH)=1234H$, et pour PUSH B : $(30FFAH)=2D5AH$.
- Pour l'instruction POP B, le contenu de la case mémoire pointée par la registre SP , est chargé dans le registre B. $((SP))=30FFAH ; (B)=2D5AH$ puis (SP) est incrémenté de deux ($(SP)= 30FFAH+2=30FFCH$).
- Enfin, pour POP A on obtient : $(A)=1234H$ et $(SP)=30FFCH + 2 = 30FFEH$.

Réponse



Réponse

- Le pointeur de pile SP pointe le haut de la pile (31000H). l'évolution du pointeur de pile durant l'exécution du programme commençant en 12E30H



Push C: C \rightarrow [SP] et SP \leftarrow SP-2

Push A: A \rightarrow [SP] et SP \leftarrow SP-2

Push B: B \rightarrow [SP] et SP \leftarrow SP-2

POPA: B \leftarrow [SP] et SP \leftarrow SP+2

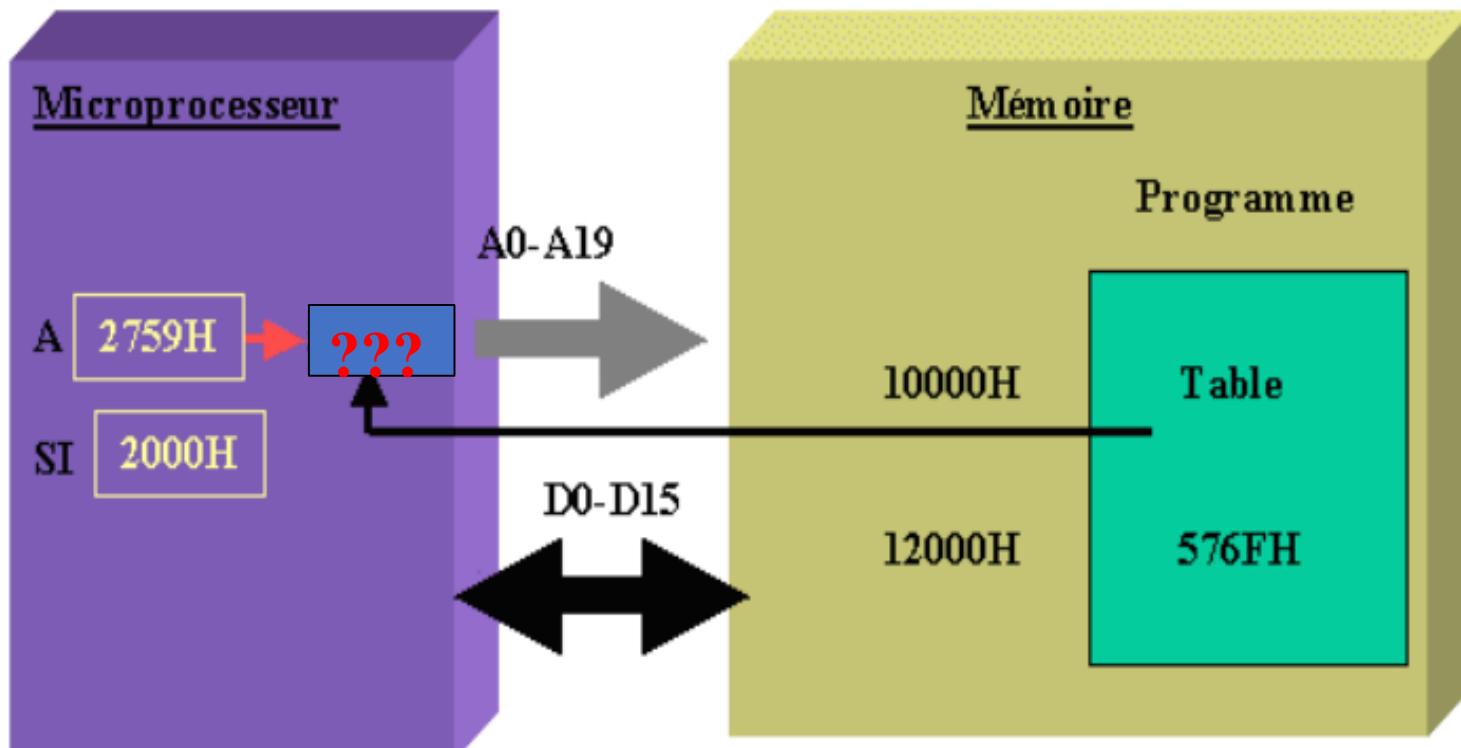
POPA: A \leftarrow [SP] et SP \leftarrow SP+2

Les registres d'index (index source SI et index destination DI)

- Le contenu de ce registre de 16 bits est une adresse.
- Les registres d'index permettent de mémoriser une adresse particulière (par exemple : début d'un tableau).
- Ces registres sont aussi utilisés pour adresser la mémoire de manière différente. C'est le mode d'adressage indexé.

Exemple :

- MOV A,[SI+10000H] place le contenu de la mémoire d'adresse 10000H+le contenu de SI, dans le registre A.



Étapes d'exécution d'une instruction

- Le traitement d'une instruction peut se découper en plusieurs phases :
 - ❖ Recherche de l'instruction (Fetch)
 - ❖ Décodage (decode)
 - ❖ Exécution (execute)

Fetch / Décodage / Exécution

Début

Lire

la prochaine instruction à exécuter
depuis la mémoire et la charger
Registre instruction (RI)

Modifier

le Compteur Ordinal pour qu'il pointe
sur la prochaine instruction à exécuter,

Décoder

l'instruction qui vient d'être chargée,

Charger

les données éventuelles dans les registres
internes,

Réaliser

l'opération,

Fin

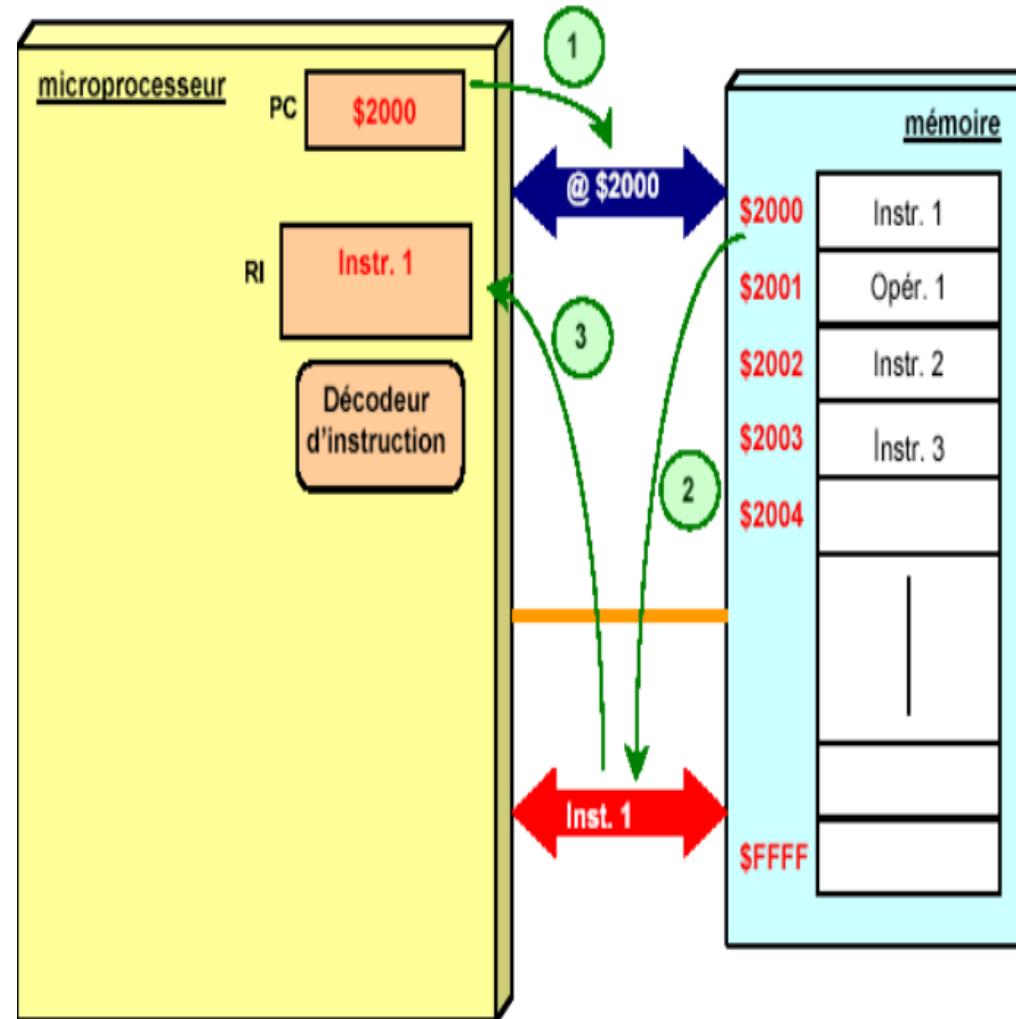
Fetch

Décodage

Exécution

Phase 1: Recherche de l'instruction à traiter

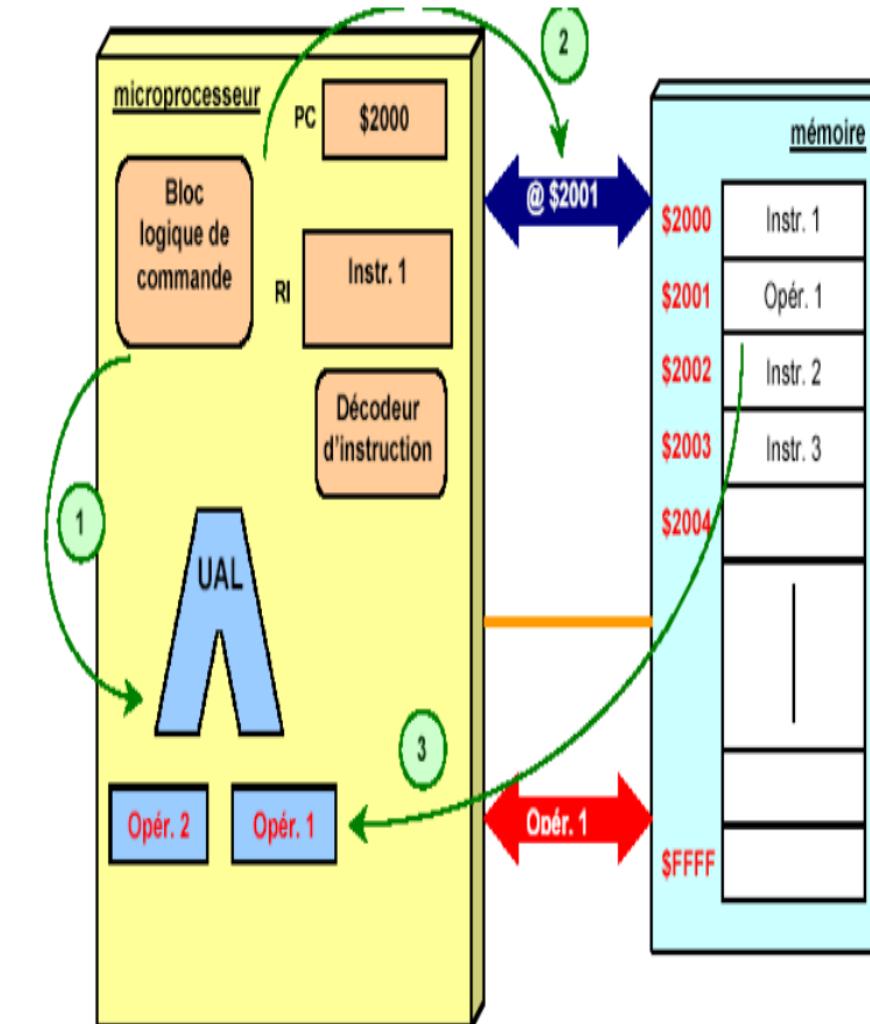
1. Le PC contient l'adresse de l'instruction suivante du programme. Cette valeur est placée sur le bus d'adresses par l'unité de commande (UC) qui émet un ordre de lecture (READ=RD=1).
2. Au bout d'un certain temps (temps d'accès à la mémoire), le contenu de la case mémoire sélectionnée est disponible sur le bus des données.
3. L'instruction est stockée dans le registre instruction du processeur.



Phase 2 : Décodage de l'instruction et recherche de l'opérande

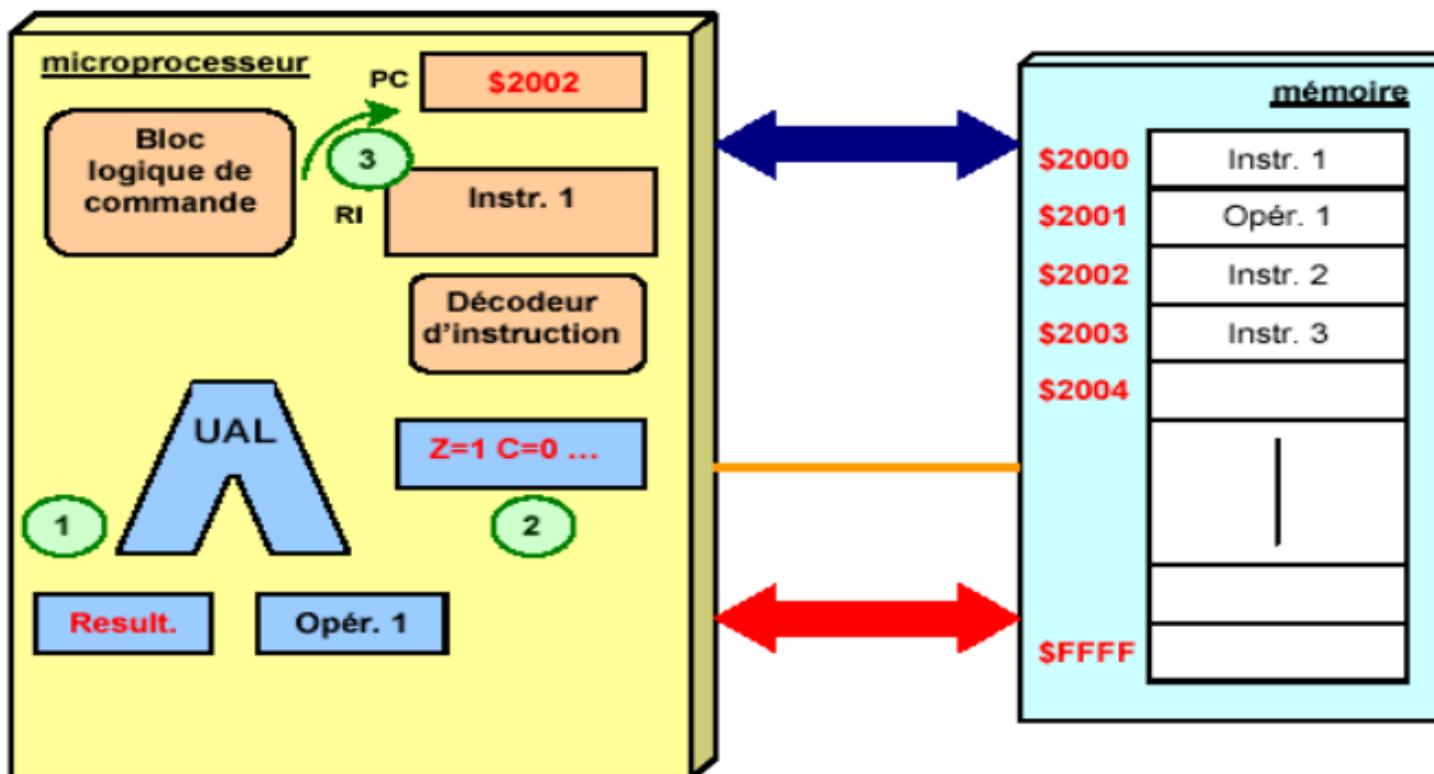
- Le registre d'instruction contient maintenant le premier mot qui contient le code opératoire et qui définit la nature de l'opération à effectuer (addition, rotation,...) et le nombre de mots de l'instruction.

1. L'unité de commande transforme l'instruction en une suite de commandes élémentaires nécessaires au traitement de l'instruction.
2. Si l'instruction nécessite une donnée en provenance de la mémoire, l'unité de commande récupère sa valeur sur le bus de données.
3. L'opérande est stocké dans un registre.



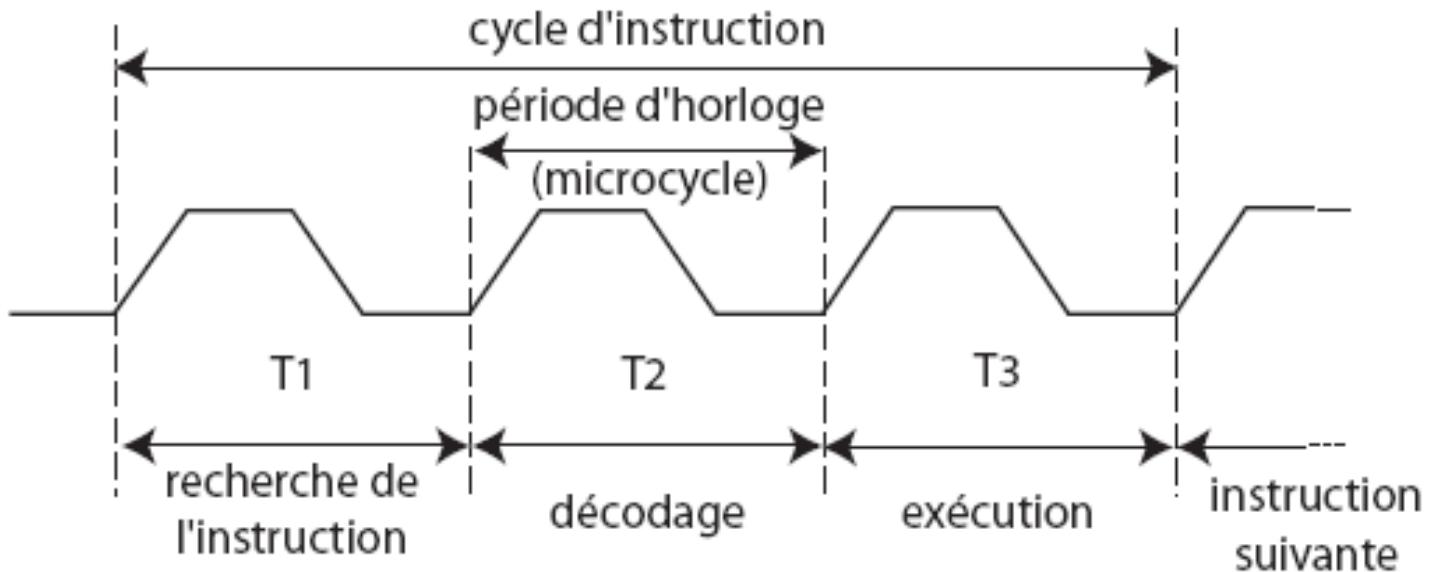
Phase 3 : Exécution de l'instruction

1. Le micro-programme réalisant l'instruction est exécuté.
2. Les drapeaux sont positionnés (registre d'état).
3. L'unité de commande positionne le PC pour l'instruction suivante.



Le signal Horloge et le cycle d'instruction

- Chaque instruction est caractérisée par le nombre de périodes d'horloge qu'elle utilise
- Exemple : horloge à 5 MHz , période $T = 1/f = 0,2 \mu\text{s}$.
- Si l'instruction s'exécute en 3 microcycles, la durée d'exécution de l'instruction est : $3 \times 0,2 = 0,6 \mu\text{s}$.

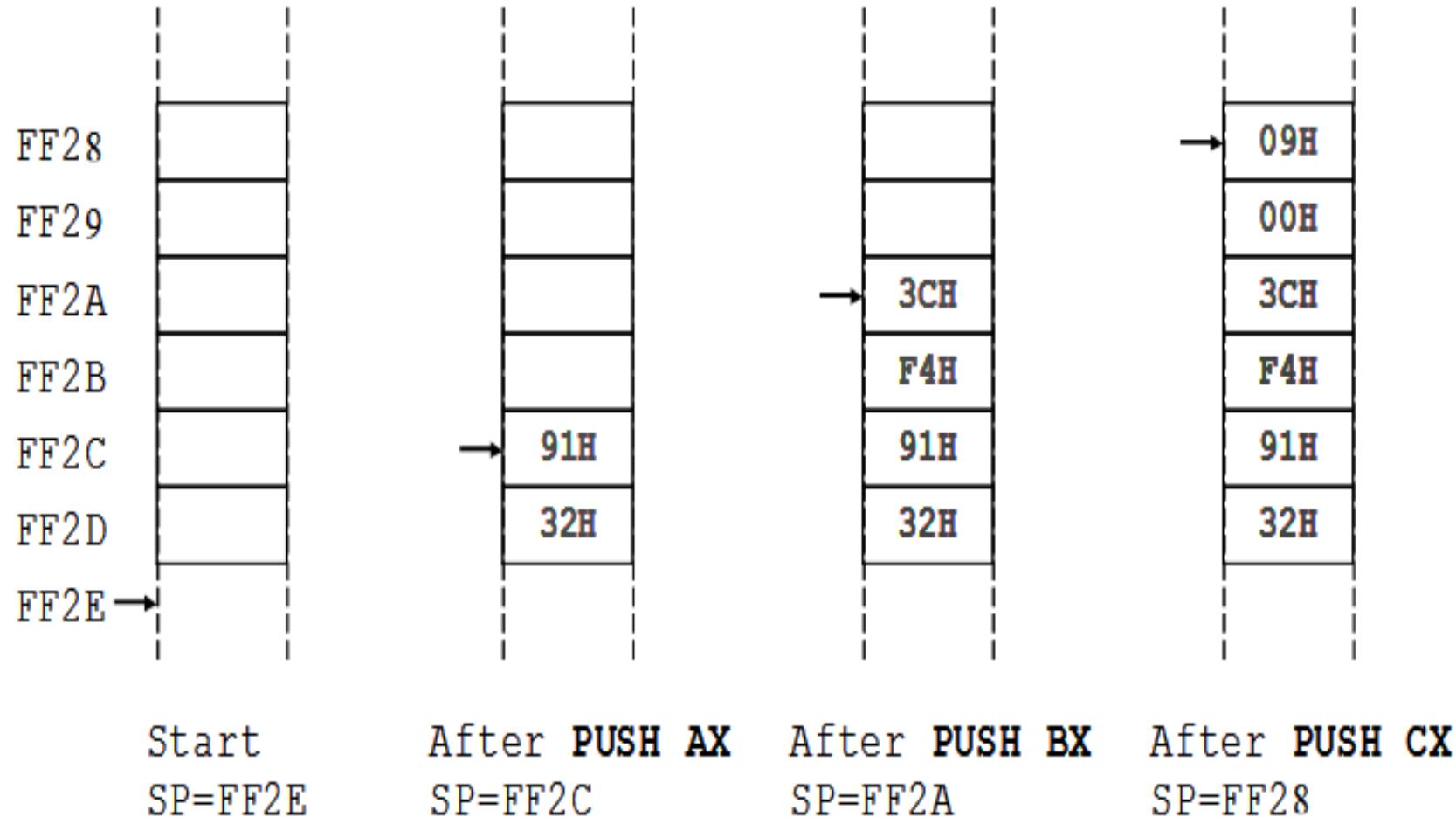


Exercice 1:

- 1) Supposons que SP = FF2EH, AX = 3291H, BX = F43CH et CX = 09. Trouver le contenu de la pile et du Pointeur de pile SP après l'exécution de chacune des instructions suivantes
 - PUSH AX
 - PUSH BX
 - PUSH CX
- 2) Trouver les contenus de POP des instructions de la question 1

Corr. ex1

- 1)



Start
SP=FF2E

After **PUSH AX**
SP=FF2C

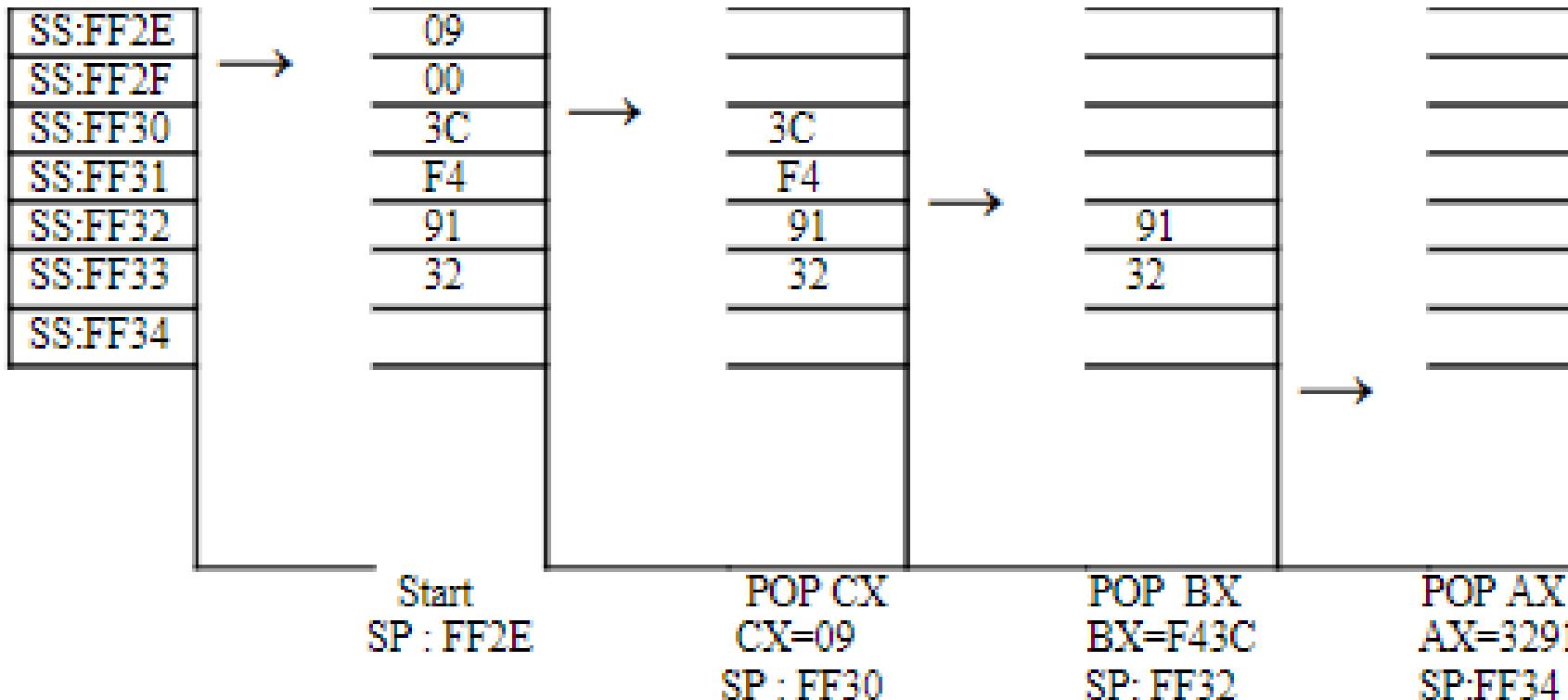
After **PUSH BX**
SP=FF2A

After **PUSH CX**
SP=FF28

Corr. ex1

- 2)

Ans: POP CX
POP BX
POP AX



Exercice 2:

- Trouver l'état des registres (F= Flag) CF, PF, AF, ZF et SF pour les opérations suivantes:

a. MOV BL, 9FH
 ADD BL, 61H

b. MOV AL, 23H
 ADD AL, 97H

c. MOV DX, 10FFH
 ADD DX, 1

Corr. Ex 2

- A. CF=1, PF=1, AF=1, ZF=1, SF=0

- B. CF=0, PF=0, AF=0, ZF=0, SF=1

- C. CF=0, PF=1, AF=1, ZF=0, SF=0

Microprocesseur 6809

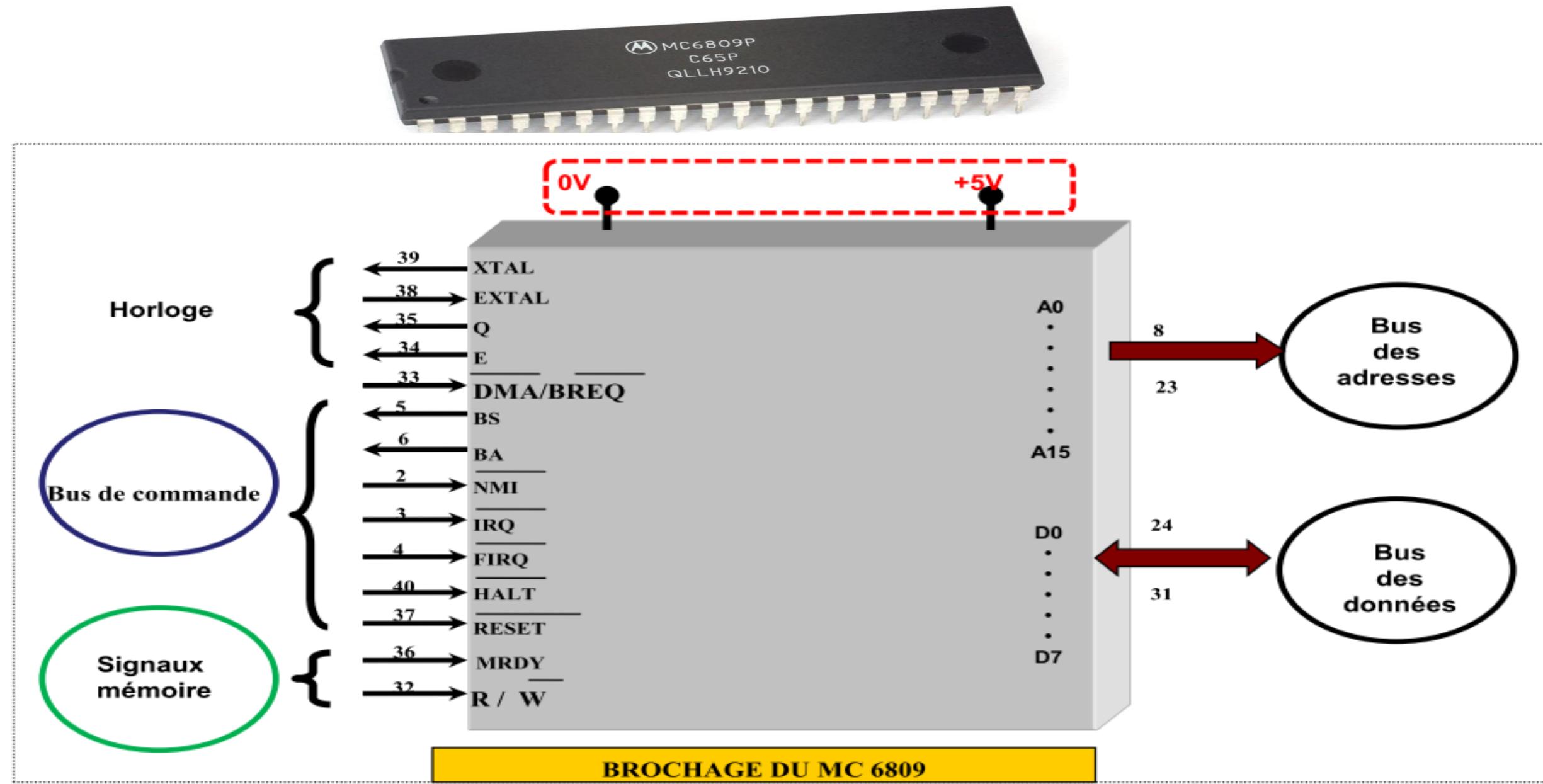


Organisation matérielle du 6809

- Le microprocesseur 6809 est un processeur 8 bits dont l'organisation interne est orientée 16 bits.
- Il est fabriqué en technologie MOS (Metal Oxide Substrate)
- Il se présente sous la forme d'un boîtier **40 broches**.



Organisation matérielle du 6809



Organisation matérielle du 6809

- NMI (No Maskable Interrupt)
- IRQ (Interrupt Request)
- FIRQ (Fast Interrupt Request)
- MRDY (Memory ready)
- DMA / BREQ (Direct Memory Acces/Bus Request).
- *EXTAL* (External clock input)
- *XTAL* (output of the crystal oscillator amplifier)
- *E* and *Q* are the clock signals. *Q* (Quadrature clock output)
- BA (Bus Available) . BS (Bus State)

Le bus des données 8 bits (D 0 à D 7)

- Ces huit broches sont bidirectionnelles. Elles permettent la communication avec le bus des données interne du microprocesseur.

Le bus des adresses 16 bits (A 0 à A 15)

- Ces broches unidirectionnelles transfèrent l'adresse 16 bits fournie par le microprocesseur au bus d'adresse du système.
- les adresses sont validées sur le front montant de Q.

Le bus de contrôle

- La broche Read/ Write: Cette broche indique le sens de transfert des données sur le bus des données.
 - R/ W = 1 lecture en cours (D 0 - D 7 sont des entrées)
 - R/ W = 0 écriture en cours (D 0 - D 7 sont des sorties)

Présentation du brochage

- **Broche d'initialisation RESET**

- Un niveau bas sur cette broche entraîne une réinitialisation complète du circuit.

Conséquences :

- l'instruction en cours est arrêté
- le registre de pagination (DP) est mis à zéro
- les interruptions IRQ et FIRQ sont masquées
- l'interruption non masquable NMI est désactivée
 - Pour être active, cette ligne doit être maintenue à un niveau bas durant un temps suffisamment long (plusieurs cycles d'horloge).

la broche : HALT (Arrêt du microprocesseur).

- Un niveau bas sur cette broche provoque l'arrêt du microprocesseur (mais à la fin de l'exécution de l'instruction en cours). Il n'y a pas perte des données. (BA = BS = 1)

Dans ce cas :

- les demandes d'interruption IRQ et FIRQ sont inhibées
- les demandes d'accès direct (DMA) à la mémoire sont autorisées.
- les demandes d'interruptions RESET et NMI sont prises en compte mais leur traitement est différé (**retardé**).

les broches d'interruption

- NMI (No Maskable Interrupt)
- IRQ (Interrupt Request)
- FIRQ (Fast Interrupt Request)
- Entrées (actives sur un niveau bas) qui peuvent interrompre le fonctionnement normal du microprocesseur sur front descendant de Q.

Présentation du brochage

- **Les lignes d'état du bus**

- BA (Bus available) et BS (Bus state): Information qui permet de connaître l'état du microprocesseur à tout moment.

BA	BS	Etat
0	0	normal
0	1	reconnaissance d'interruption
1	0	reconnaissance de synchronisation externe
1	1	arrêt bus disponible

Présentation du brochage

- **1er cas :**

Le microprocesseur est en fonctionnement normal, il gère les bus d'adresses et de données.

- **2ème cas :**

le microprocesseur est en phase de reconnaissance d'interruption pendant deux cycles. Cet état correspond à la recherche des vecteurs d'interruption : Reset, NMI, IRQ, SW1,2 et 3. (SW=Software)

- 3ème cas :**

Ce signal apparaît lorsque le microprocesseur rencontre l'instruction de synchronisation externe (niveau bas sur SYNC). Il attend alors cette synchronisation sur une des lignes d'interruption. Les bus sont en haute impédance pendant ce temps.

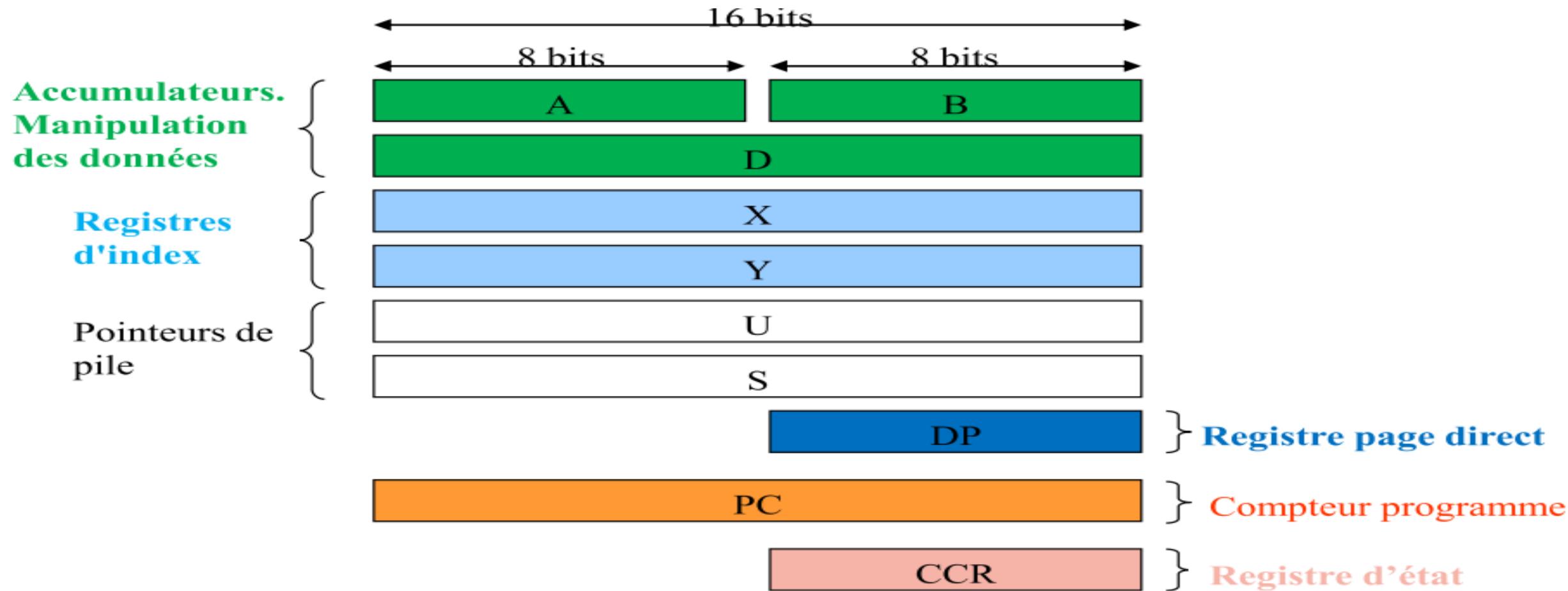
- Dernier cas :**

Correspond à l'arrêt du microprocesseur (niveau bas sur HALT).

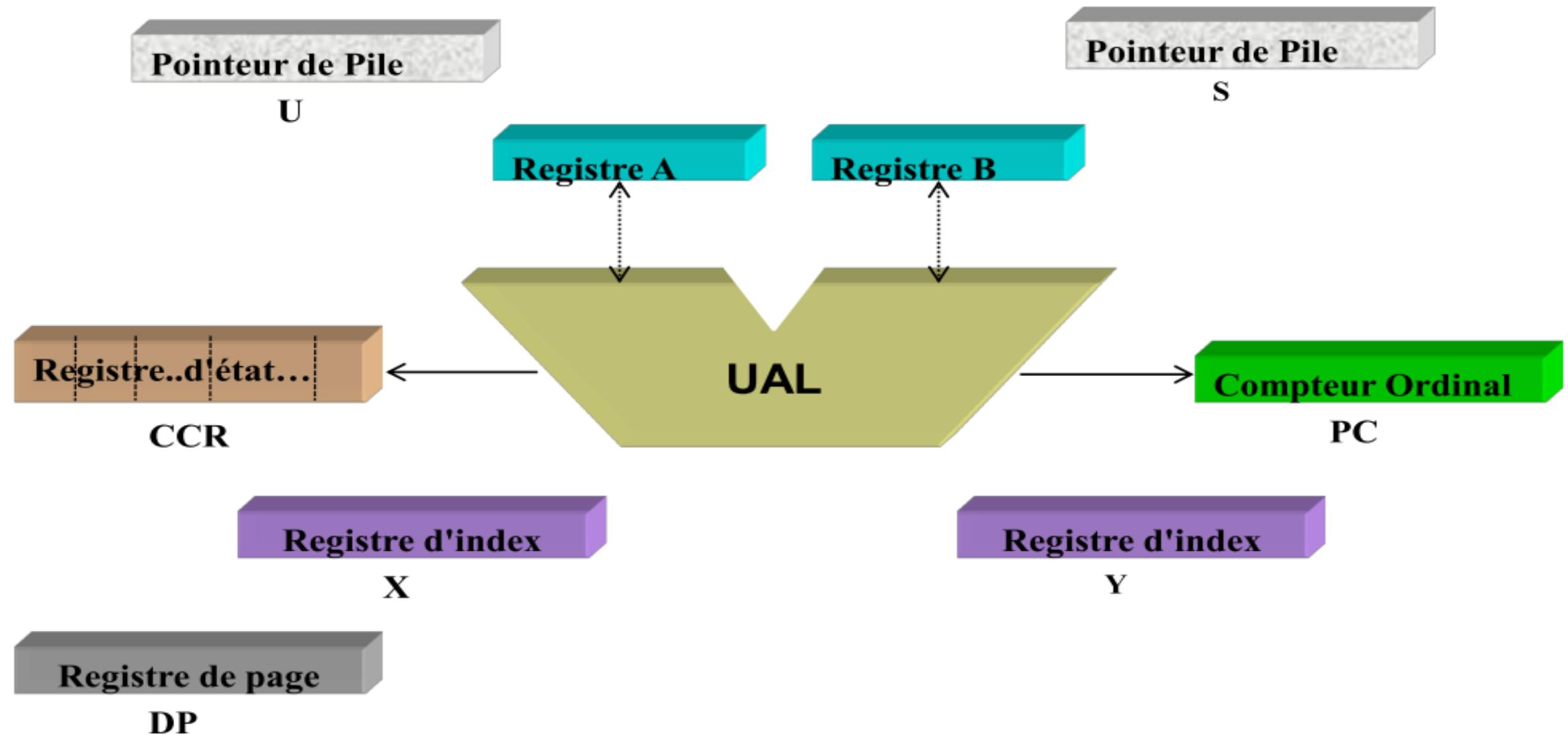
Le microprocesseur laisse la gestion des bus des données et des adresses à un circuit annexe (contrôleur de DMA). Les bus sont en haute impédance. La ligne BA au niveau haut indique que les bus sont en haute impédance.

Architecture 6809

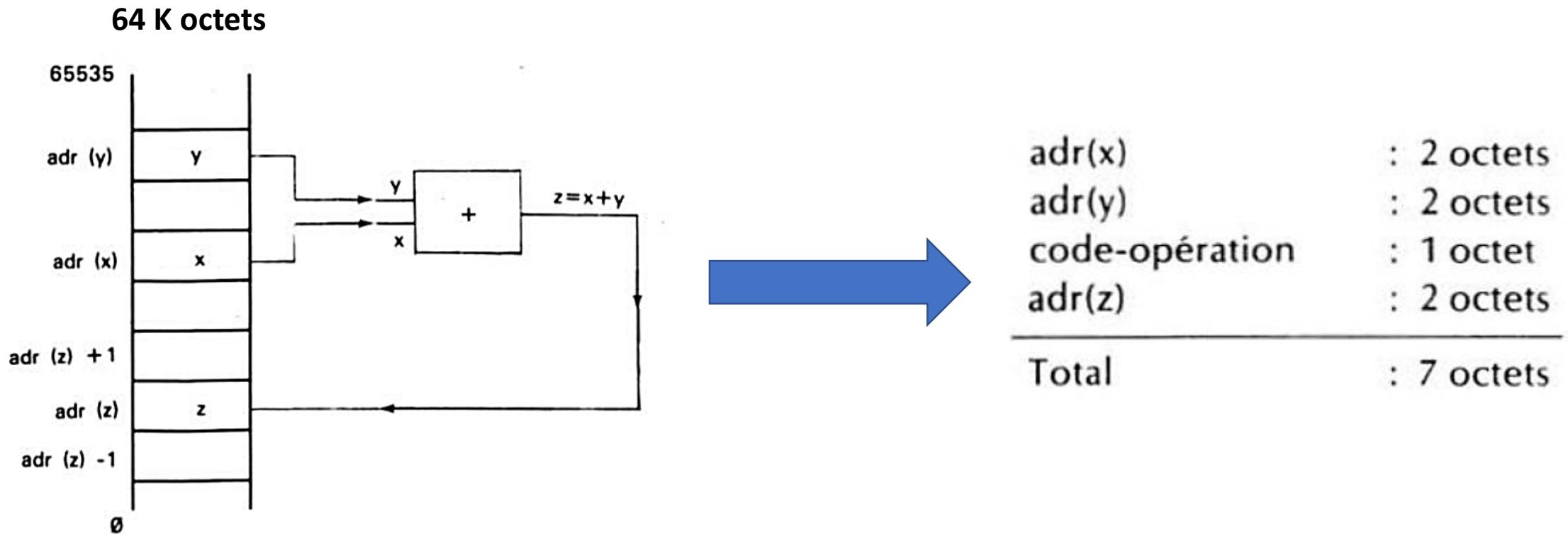
- Le microprocesseur 6809 comporte NEUF registres internes programmables accessibles par l'utilisateur:



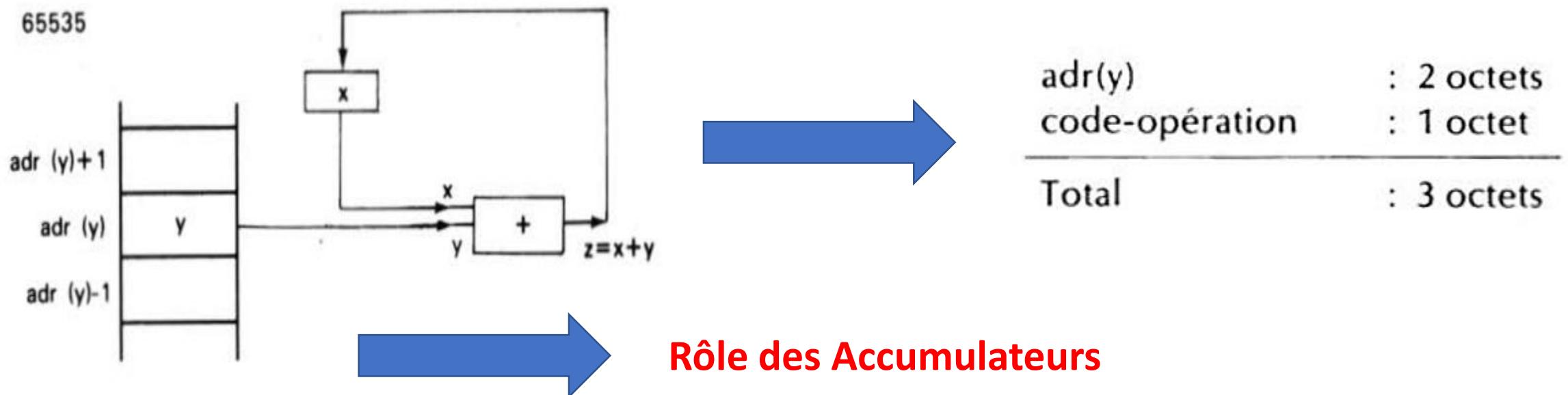
Architecture 6809



- **Exemple: addition de deux nombres x et y: $z = x + y$**
- Nous allons supposer que x et y sont situés dans deux cases-mémoire distinctes donc à deux adresses différentes que nous appellerons $adr(x)$ et $adr(y)$. Le résultat de l'addition, soit z, sera stocké à l'adresse $adr(z)$.

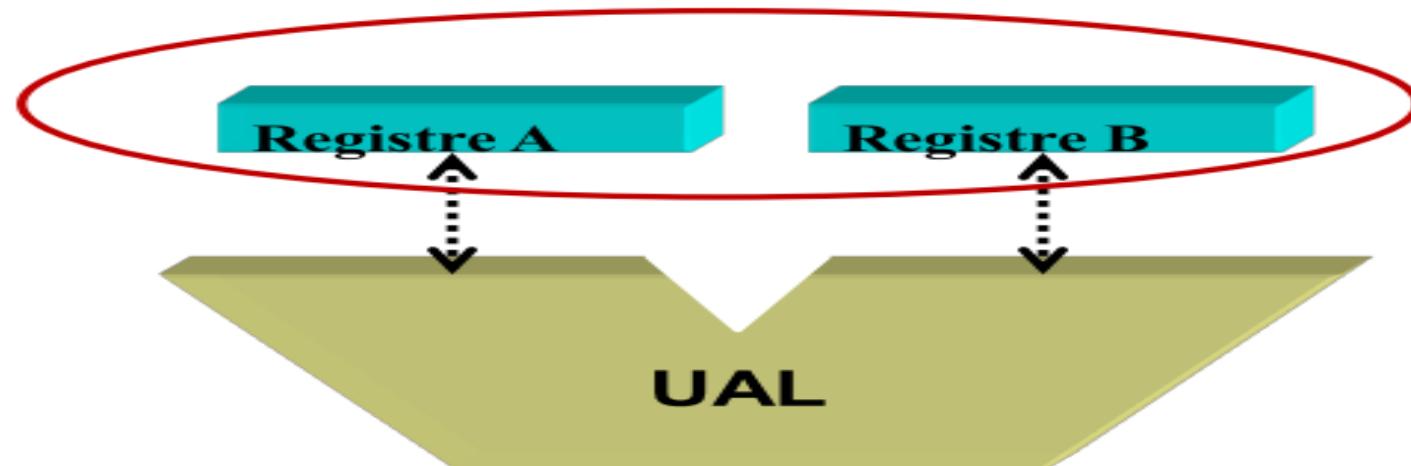


- **Exemple: addition de deux nombres x et y: $z = x + y$**
- Dans ce cas le contenu de l'adresse de y, ($adr(y)$) est ajouté à x contenu dans une case-mémoire particulière et située en dehors de l'espace mémoire adressable du microprocesseur. Le résultat $z = x + y$ est ensuite mis dans cette même case-mémoire.



- **Accumulateurs : A, B et D**

- Les calculs arithmétiques et les instructions de comparaison ainsi que les manipulations de données se font grâce aux accumulateurs A et B.
- Ces deux registres sont interchangeables sauf pour quelques instructions (ABX, DAA) et les opérations sur 16 bits.
 - ABX : Addition de l'accumulateur B à X(Non signé)
 - DAA : Ajustement décimal de l'accumulateur A
- $D = A \text{ } B$ (Concaténation de A et B->16bits): A est l'octet de poids fort.



- **Registres d'index : X,Y (registres de 16 bits)**

- Ces deux pointeurs d'utilisation parfaitement identique sont utilisés dans les modes d'adressage indexé.
- Sont utilisés aussi:
 - comme registres de stockage de résultats intermédiaires
 - comme compteurs de boucle.
- Les données - 16 bits- contenues dans ces registres servent de pointeur de données (adresses).
- Ces adresses "peuvent être modifiées" par une constante, prise comme valeur de déplacement (offset) qui permet alors de calculer une adresse effective.

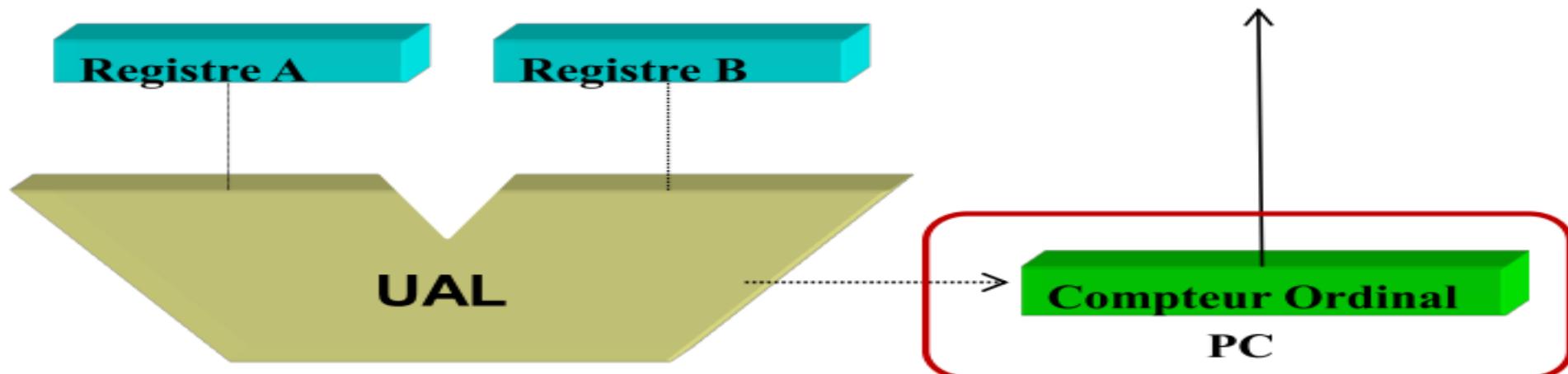


- **Pointeurs de pile: S,U (registres 16 bits)**

- le pointeur de pile **S** (Système) est utilisé automatiquement par le microprocesseur pour mémoriser l'état de tous ses registres internes dans le cas où il doit exécuter un sous programme (d'interruption ou non).
- le pointeur de pile **U** (Utilisateur) est géré exclusivement par le programmeur pour effectuer, avec facilité, le passage des paramètres entre les programmes et les sous programmes.
 - Les registre U et S peuvent faire office de pointeurs - registres d'index.

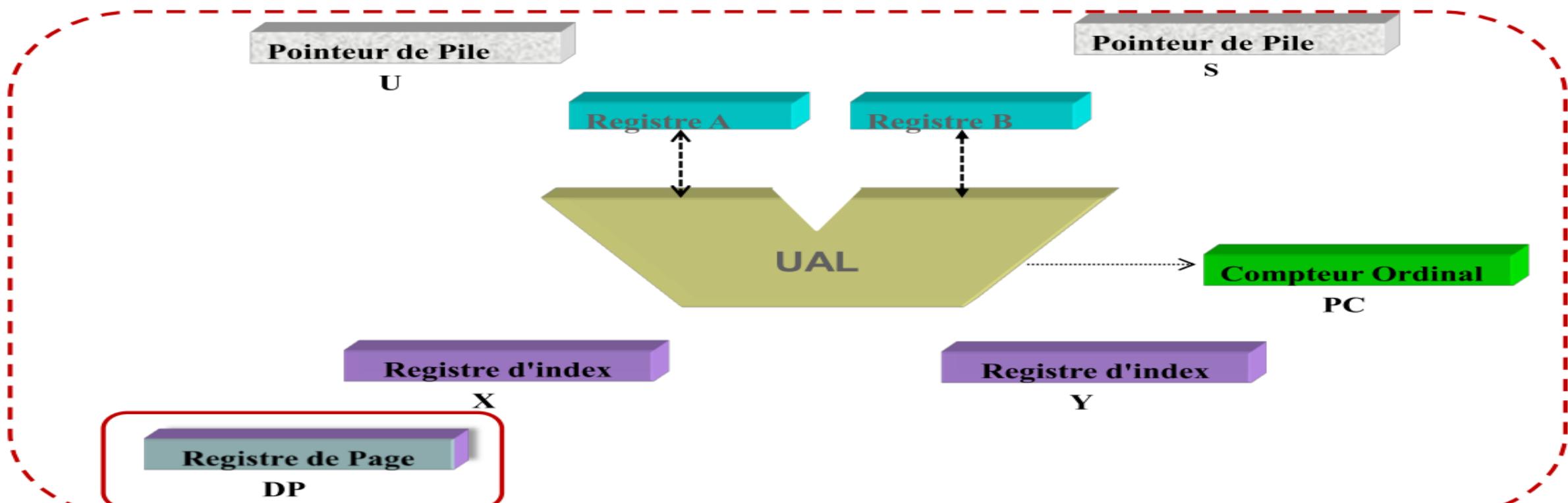
- **Registre compteur programme PC**

- Le contenu de ce registre (16 bits) détermine l'adresse de l'instruction qui doit être exécuté par le processeur.
- Il pointe en permanence l'adresse de la prochaine instruction à exécuter.
- Dans certain cas, ce compteur programme peut être utilisé comme un index.



- **Registre de page: DP**

- Ce registre (8 bits) est prévu pour étendre les possibilités d'adressage direct à tout l'espace mémoire, sous contrôle du logiciel.



- **Registre de codes condition: CCR**

- Le registre codes condition (8 bits) définit à tout instant l'état des indicateurs du processeur.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
E	F	H	I	N	Z	V	C

Architecture 6809

C	Retenue (Carry)	Positionné lors d'une opération arithmétique
V	Dépassement (Overflow)	Positionné si le résultat(en complément à deux) d'une opération arithmétique déborde
Z	Zéro	Mis à 1 si le résultat de l'opération précédente est nul
N	Négatif	Indique un résultat négatif.
I	Masque d'interruptions IRQ	Lorsqu'il est à 1, masque les interruptions IRQ
H	Demi-retenue	
F	Masque d'interruptions FIRQ	Lorsqu'il est à 1, masque les interruptions FIRQ
E	Etat de sauvegarde	Si à 1, tout le contexte du processeur est sauvegardé dans la pile

MODE D'ADRESSAGE ET PROGRAMMATION EN ASSEMBLEUR

Mode d'adressage

- Un mode d'adressage est **un moyen d'accéder** à une case mémoire donnée.
- **Exemple: (addition de deux nombres z=x+y)**
 - Pour connaître un nombre x, il fallait spécifier son adresse sur 16 bits (ou deux octets).
→ Le fait de définir x par la donnée de son adresse constitue **un mode d'adressage**
- Le MPU (Microprocessor Unit) 6809 possède:
 - **59 instructions** de base
 - NEUF modes d'adressage (**9 façons de coder les adresses**)
 - **1 464** possibilités d'instructions

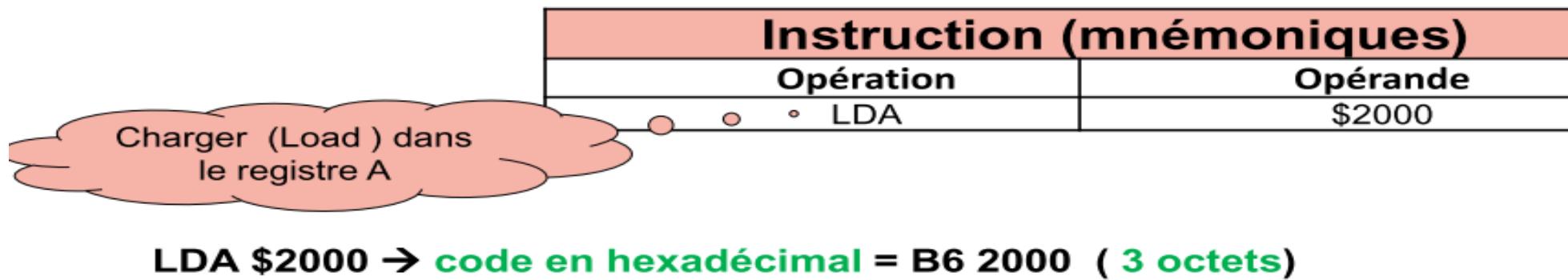
=> Le µp6809 sait automatiquement quelle est l'opération à effectuer ainsi que les registres concernés.

Mode d'adressage

- Au moyen des **signaux** qu'il génère sur le bus d'adresses, le microprocesseur a la possibilité **d'adresser les divers circuits mémoires et interfaces**, qui lui sont connectés au travers des bus afin d'accéder à leur contenu.
- Cette accès se traduit par une **opération d'adressage**
- Cette opération peut se faire de plusieurs façons grâce à la présence de **différents modes d'adressage**.
- **Remarque** : La puissance d'un microprocesseur dépend de son **jeu d'instructions** et aussi des ses **modes d'adressage**.

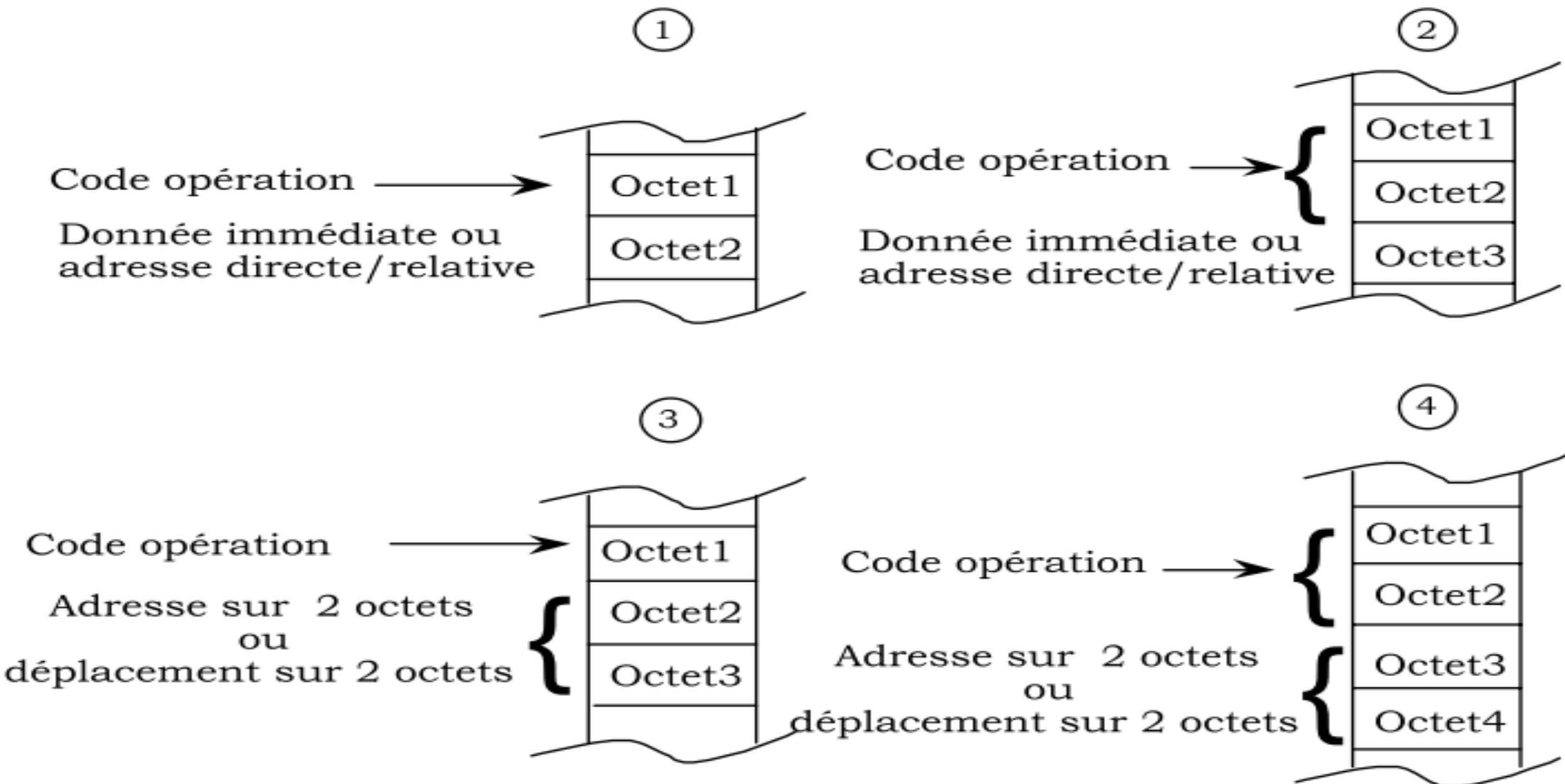
Structure d'une instruction

- Elle comporte de un à 5 octets (dépend du mode d'adressage).
 - Le premier (parfois le deuxième) octet indique l'action à effectuer
→ **correspond au code de l'instruction**
 - Les octets suivants précisent les opérandes ou sur quelques registres cette action agira.



- Dans le champ opérande on peut trouver :
 - des nombres,
 - des noms de variables,
 - des étiquettes,
 - des expressions arithmétiques ou logiques.

Structure d'une instruction



Modes d'adressage

- Les modes d'adressage sont :
 - l'adressage **inhérent** ou implicite
 - l'adressage **immédiat**
 - l'adressage **étendu** (direct)
 - l'adressage étendu indirect
 - l'adressage **direct**
 - l'adressage par registre
 - l'adressage **indexé** direct
 - l'adressage indexé indirect
 - l'adressage **relatif**

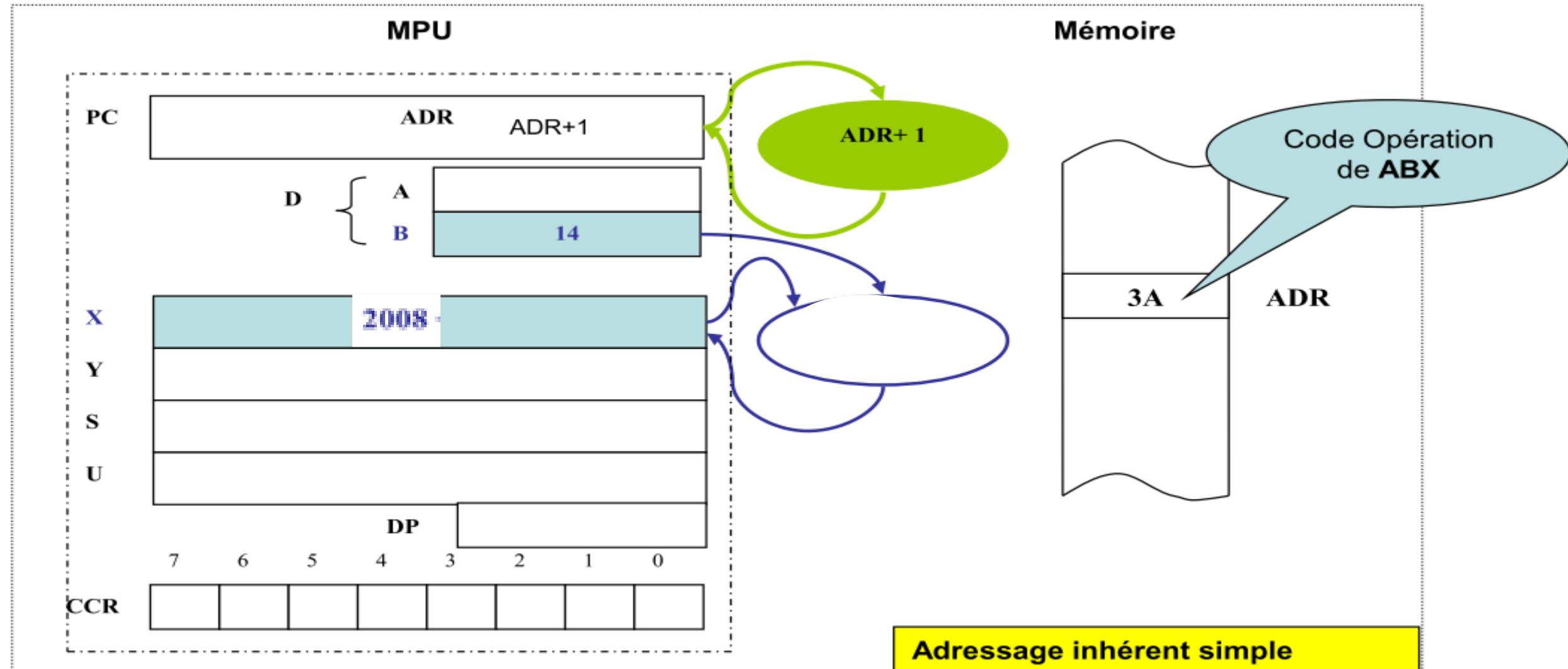
1.Mode d'adressage inhérent

L'adressage inhérent est utilisé par les instructions qui agissent sur les **registres internes du MPU** et non sur la mémoire. Il existe deux types de mode d'adressage inhérent :

❖ Adressage inhérent simple

- Le code opération contient toute l'information nécessaire à l'exécution de l'instruction.
- Ces instructions codées sur un octet sont: **ABX, INCB, NOP.....**
- **Exemple:** ABX → addition de l'accumulateur B à l'index X.

1.Mode d'adressage inhérent



1.Mode d'adressage inhérent

❖ Adressage inhérent paramétré

L'instruction comporte un octet supplémentaire permettant de préciser les opérandes intervenant dans l'instruction. La présence de cet octet supplémentaire est indispensable pour les instructions de type:

- Échange et transfert de registres : **EXG, TFR**
- Instructions d'accès aux piles: **PSHS, PSHU, PULS, PULU**
- Attente d'interruption : **CWAI** (Clear WAit Interrupt)

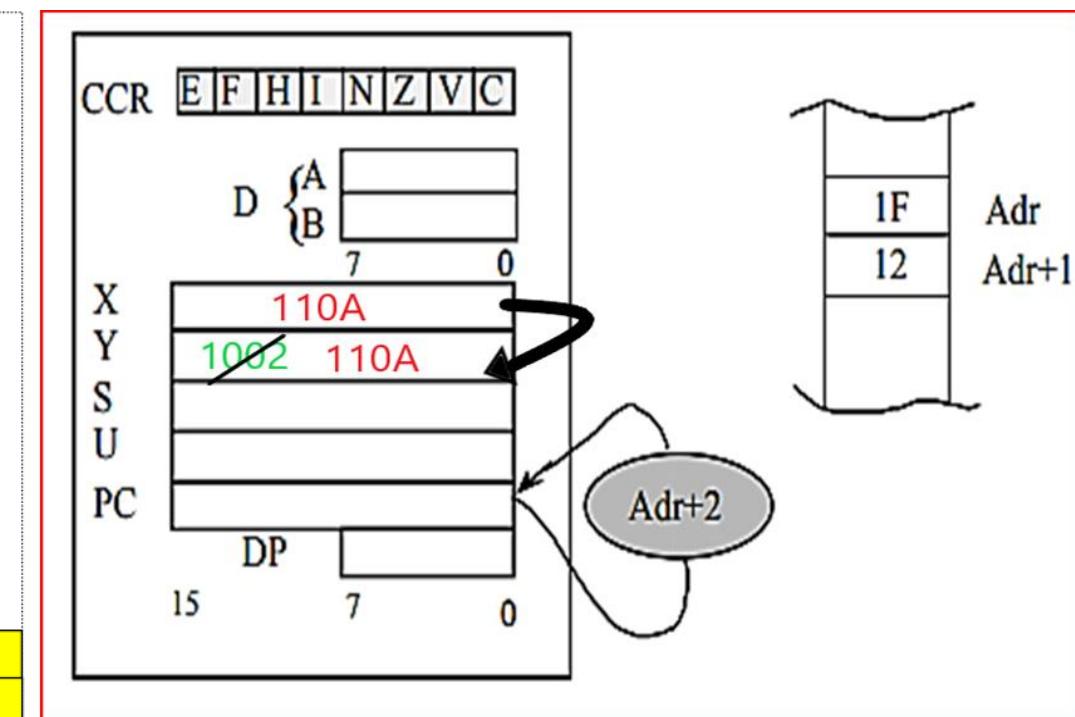
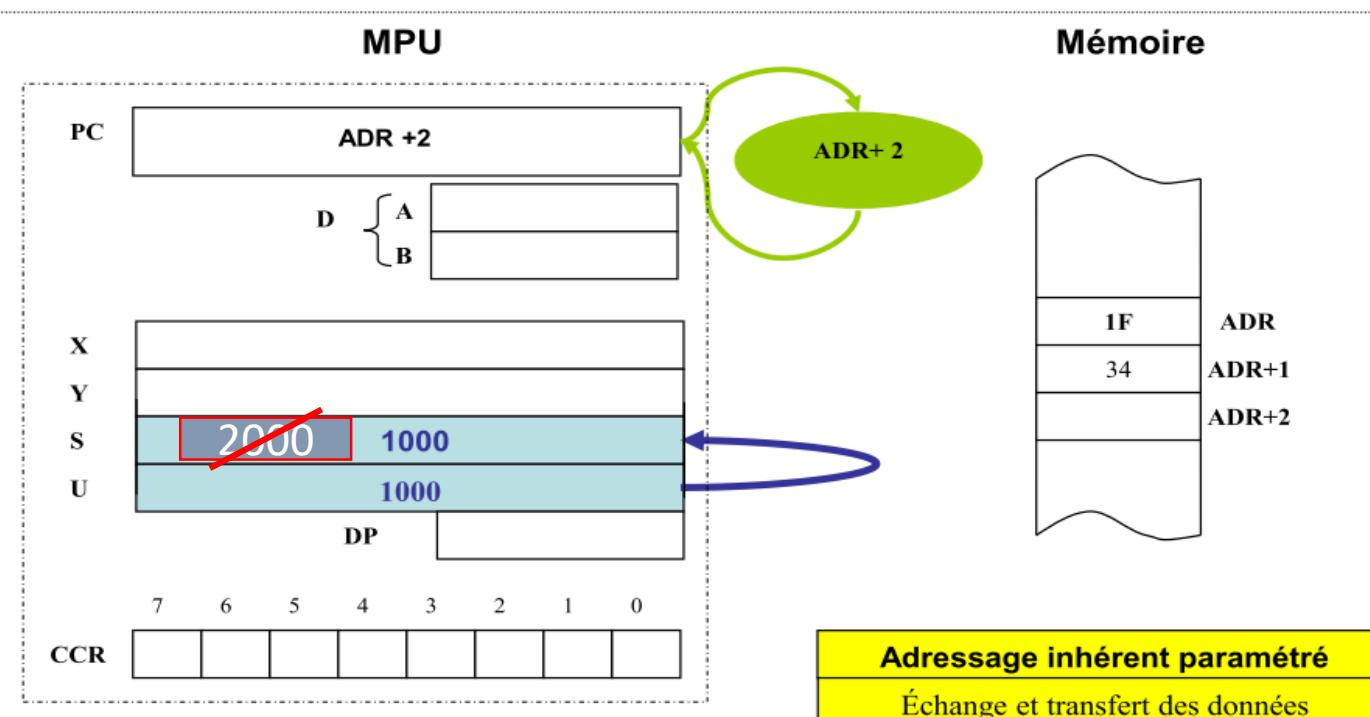
1. Mode d'adressage inhérent

■ Échange et transfert de registres:

- le premier octet détermine le code opération pur,
- le second détermine les registres source et destination:

■ Exemple : TFR U,S → transfert de U dans S

■ Exemple : TFR X,Y → transfert de X dans Y



1.Mode d'adressage inhérent

Post-octet transfert/échange :

code	Registre
0000	D
0001	X
0010	Y
0011	U
0100	S
0101	PC
1000	A
1001	B
1010	CCR
1011	DP

Code Opération
1F

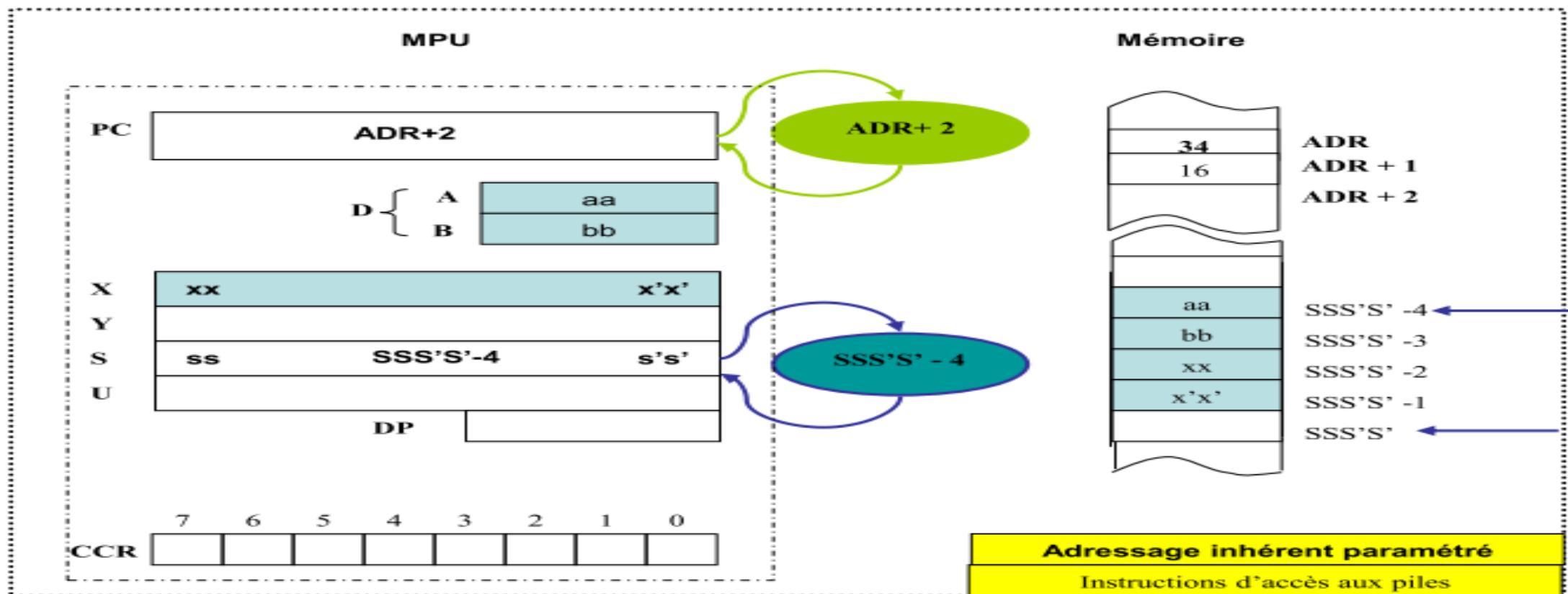
TFR **U,S** → transfert de **U** dans **S**

Post-Octet
34

1. Mode d'adressage inhérent

▪ Instructions d'accès aux piles:

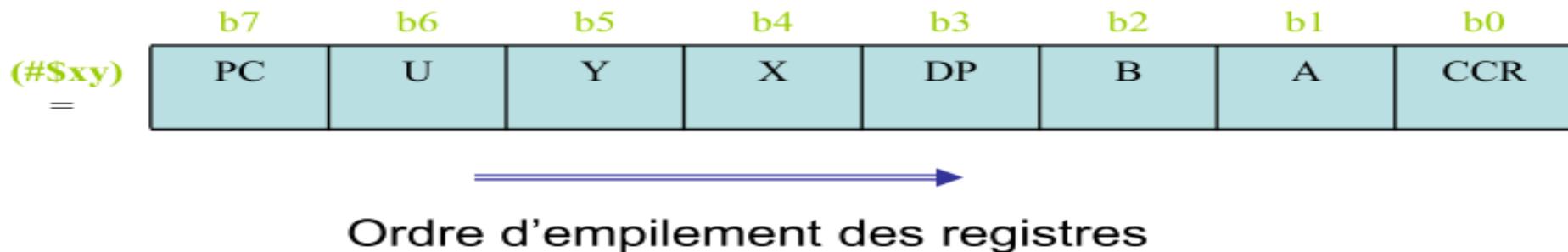
- le premier octet détermine le code opératoire pur,
 - le post-octet détermine les registres concernés par l'accès à la pile.
- Exemple: **PSHS A, B, X** → sauvegarde de A, B, X dans la pile



1.Mode d'adressage inhérent

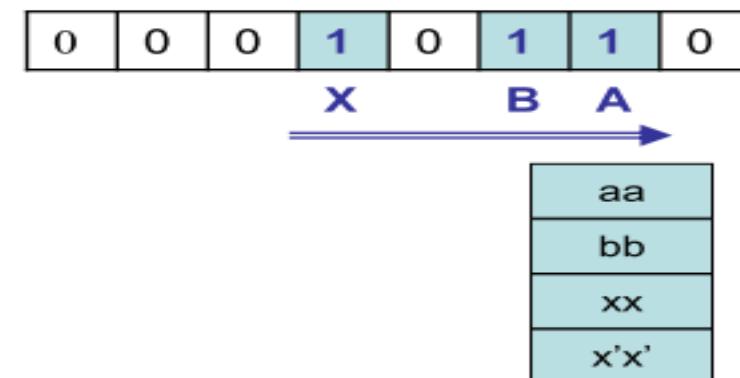
• Mnémoniques d'empilement

- **PSHS** liste explicite de registres à empiler ou (implicitement)
- **PSHS #\$xy**



Si b_i ($0 \leq i \leq 7$) = 1, alors le registre correspondant est empilé dans la pile S. Ainsi la donnée immédiate **#\$xy** permet de sélectionner le(s) registre(s) à sauvegarder.

Exemple: **PSHS A,B,X** \Leftrightarrow **PSHS #\$16**



1.Mode d'adressage inhérent

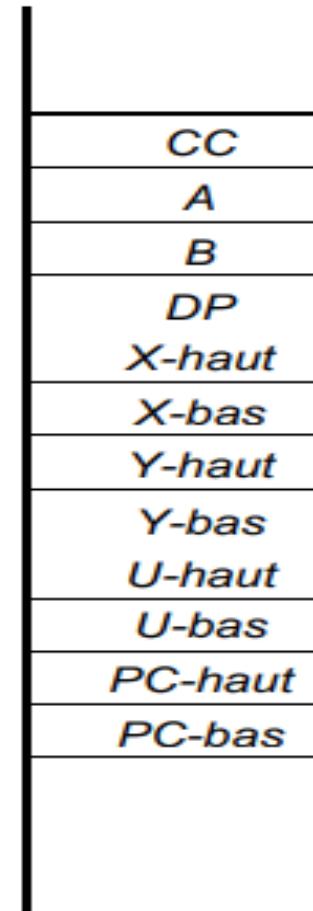
- Mnémoniques d'empilement

- **PSHS** liste explicite de registres à empiler ou (implicitement)
- **Exemple: PSHS #\$FF**

ssss - C ssss - B ssss - A ssss - 9
ssss - 8
ssss - 7
ssss - 6

ssss - 5
ssss - 4

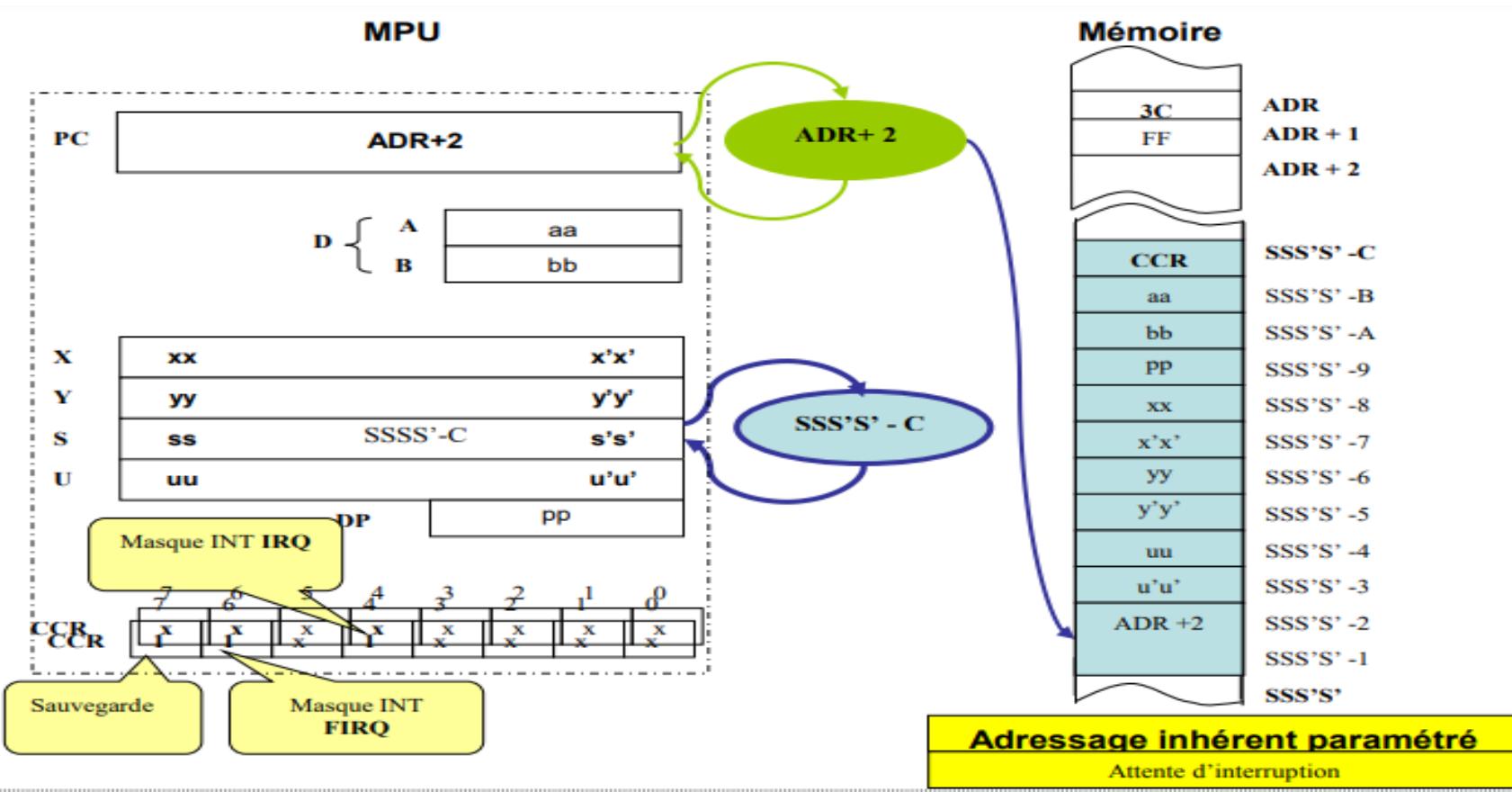
ssss - 3
ssss - 2
ssss - 1
ssss



1. Mode d'adressage inhérent

■ Attente d'interruption:

- le premier octet est associé à l'instruction CWAI, le second sert à masquer ou à valider les interruptions.
- Exemple : CWAI # \$ FF → attente d'interruption



2. Mode d'adressage immédiat

- Dans ce mode d'adressage, le **code opératoire** 8 bit est suivi d'une valeur qui est **l'opérande** de l'instruction.
- Ce type d'adressage permet de charger les registres internes du microprocesseur avec la **valeur de l'opérande**.
- Le symbole « # » signifie **immédiat** dans la syntaxe assembleur.
- Il existe trois types d'instructions dans ce mode d'adressage :
 - ❖ Instructions sur deux octets
 - ❖ Instructions sur trois octets
 - ❖ Instructions sur quatre octets

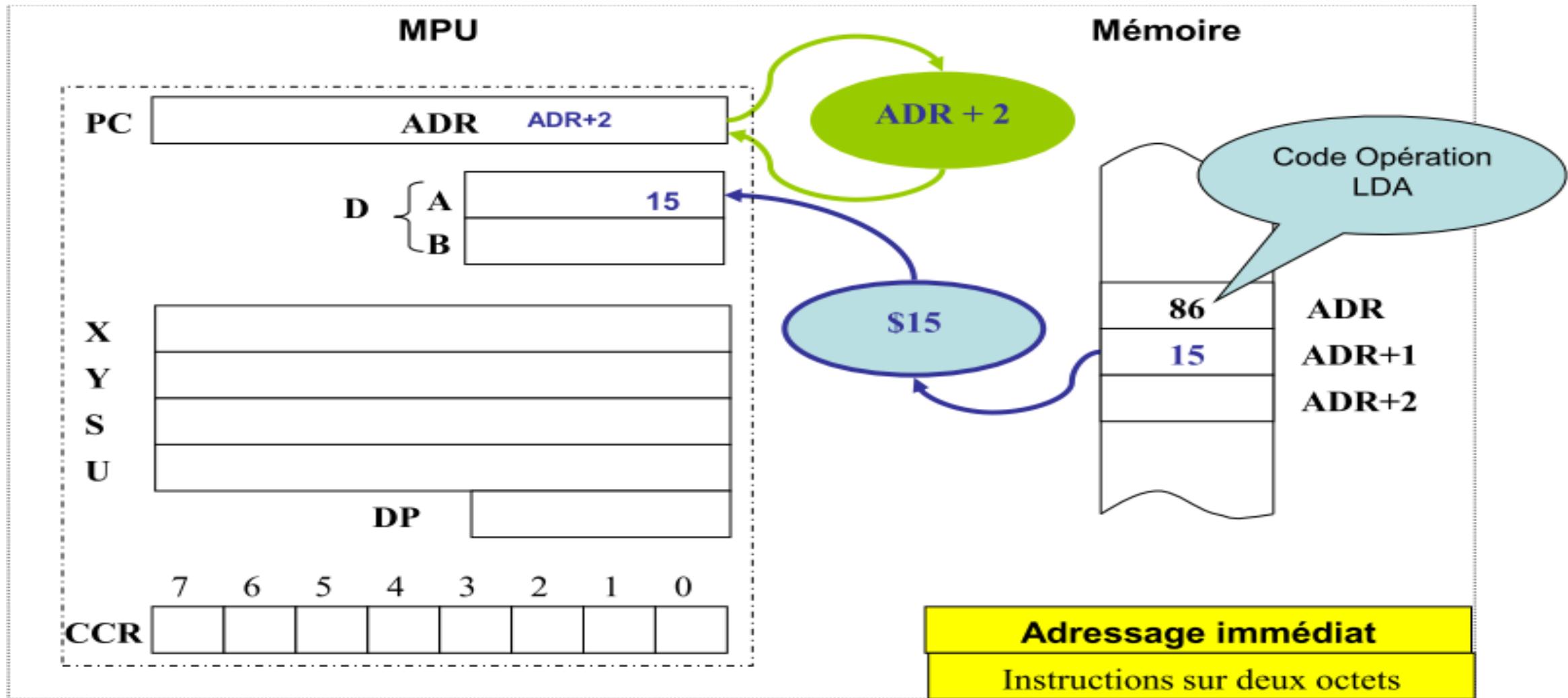
2. Mode d'adressage immédiat

- **Instructions sur deux octets:**

- Le premier octet contient le code opératoire,
- le second contient la constante 8 bits.
- Ce type d'instruction est réservé pour travailler sur les registres 8 bits du microprocesseur
- **Exemple :** LDA #\$15 → charger la valeur \$15 dans l'accumulateur A. Le premier octet contient le code opératoire

2. Mode d'adressage immédiat

- Exemple : LDA #\$15 → charger la valeur \$15 dans l'accumulateur A. Le premier octet contient le code opératoire



2. Mode d'adressage immédiat

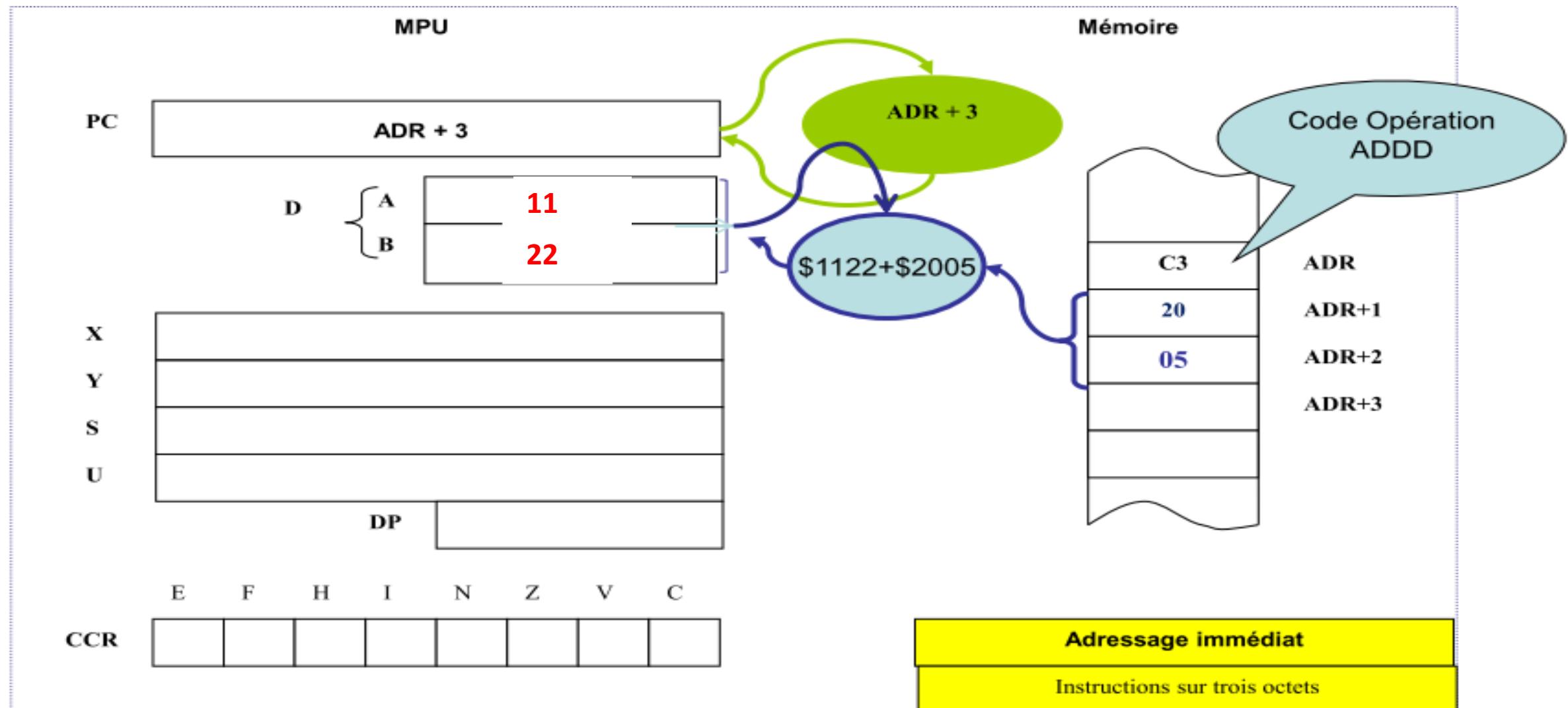
▪ Instructions sur trois octets

- Le premier octet contient le code opératoire,
- le second et le troisième contiennent la constante 16 bits.
- Ce type d'instructions est réservé pour travailler sur les registres 16 bits du microprocesseur.

▪ Exemple : ADDD #\$2005 → addition du contenu de l'accumulateur D et de \$2005, le résultat se trouve dans D.

2. Mode d'adressage immédiat

- Exemple : ADDD #\$2005 → addition du contenu de l'accumulateur D et de \$2005, le résultat se trouve dans D.



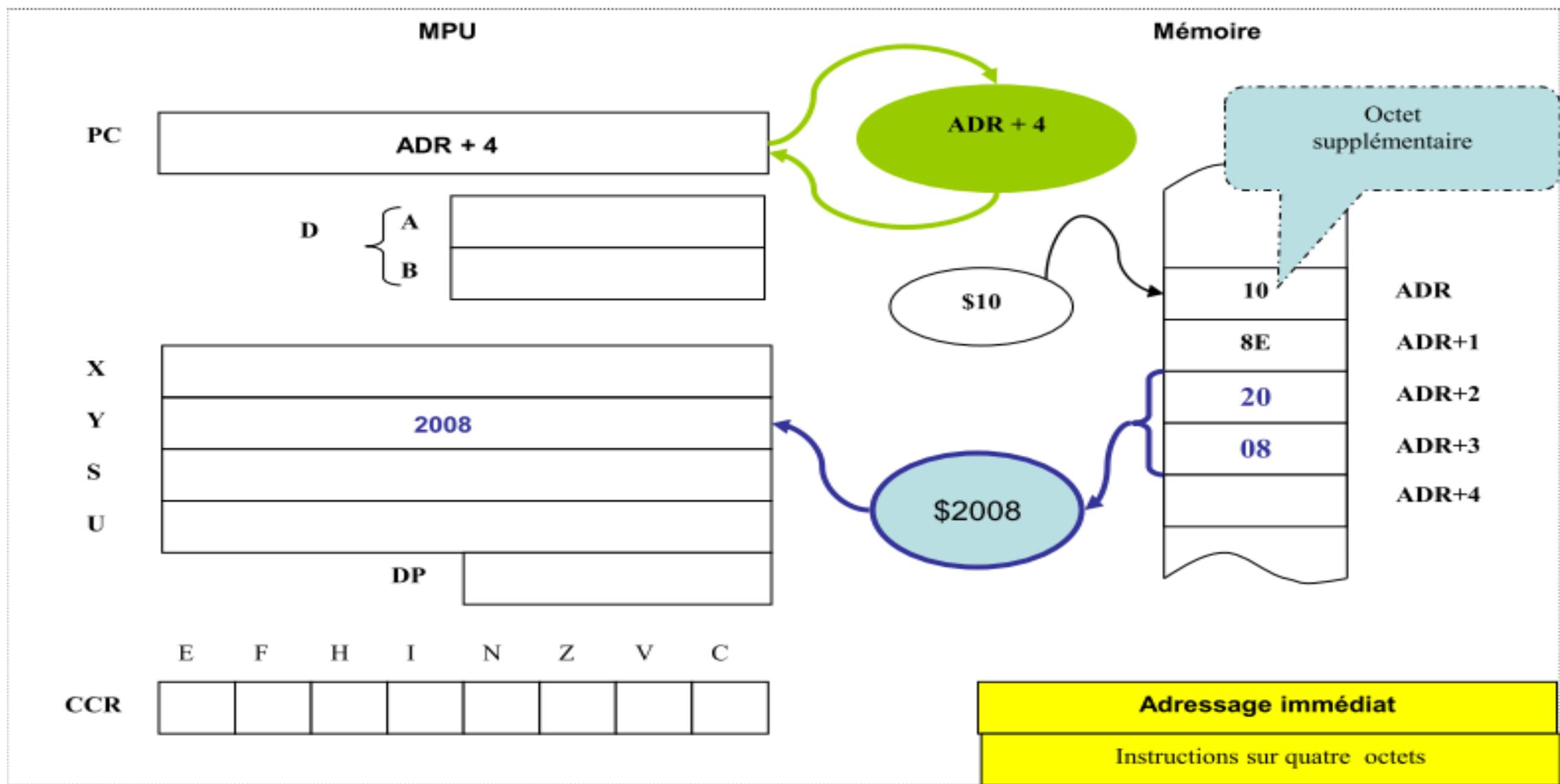
2. Mode d'adressage immédiat

- **Instructions sur quatre octets**

- Dans ce type d'instruction, le code opératoire utilise deux octets mémoire, la constante également.
 - Le **premier octet (\$10) (préfixe)** est nécessaire pour les instructions : CMPD, CMPS, CMPU, CMPY, LDS, LDY, STY, STS.
-
- Exemple: LDY #\$2008 → charger l'index Y avec la valeur \$2008

2. Mode d'adressage immédiat

- Exemple: LDY #\$2008 → charger l'index Y avec la valeur \$2008



3. Mode d'adressage direct

- On exprime le lieu de l'action par l'expression de l'adresse effective. Le **code opérande** indique la **partie basse** de cette adresse. La partie haute de l'adresse est fournie par le contenu du **Registre Direct de Page (DP)**.
- **Intérêt** : Ce mode nécessite moins de place mémoire (1 octet donc taille mémoire **réduite**) par conséquent l'exécution de l'instruction est plus rapide.
- **Remarque** : Avec ce mode, la mémoire est découpée en 256 pages de 256 octets chacune.
- Ce mode est intéressant dans le cadre des systèmes d'exploitation **temps réel multitâche**, où on alloue à chaque tâche une page.

3. Mode d'adressage direct

- ❖ Le code opératoire (un ou 2 octet)
- ❖ L'opérande (1 octet) : **Poids Faible** (8 bits) de l'**A**dresse **E**ffective



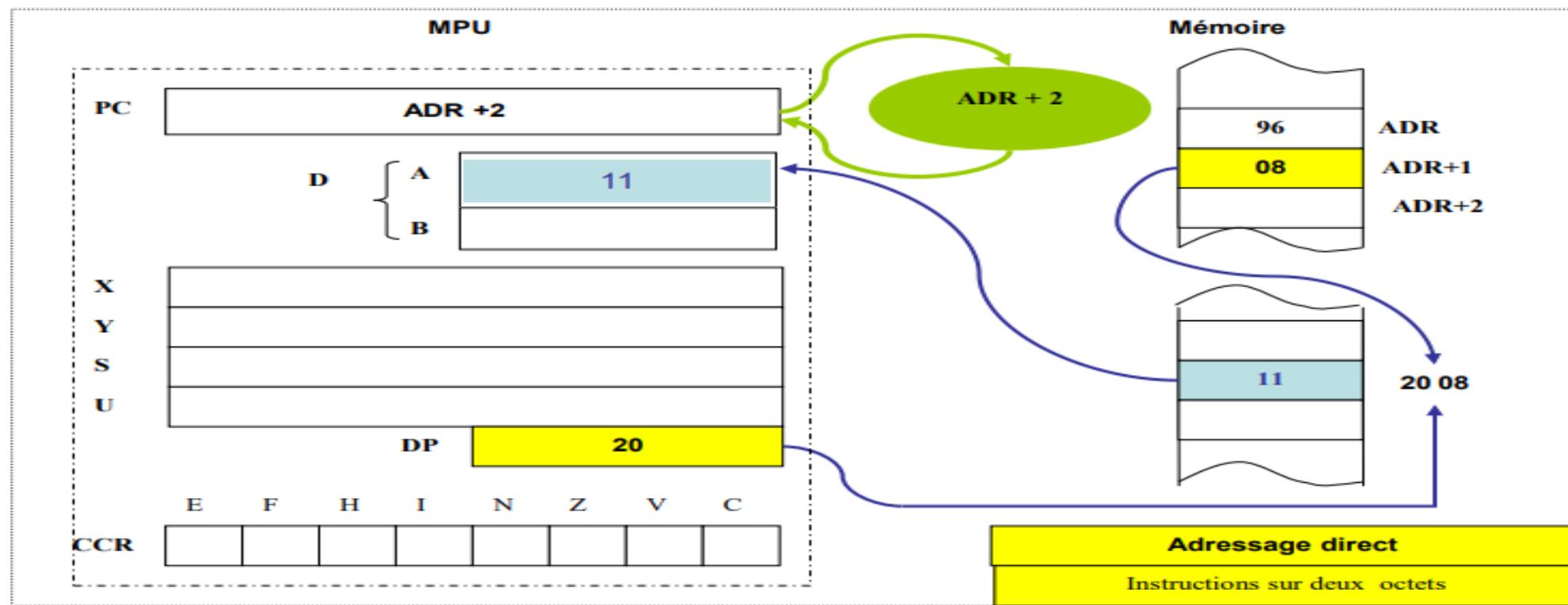
Le symbole « < » est une directive assembleur qui force l'adressage direct. Il existe deux types d'instructions dans ce mode d'adressage :

- **Instructions sur deux octets**
- **Instructions sur trois octets**

3. Mode d'adressage direct

▪ Instructions sur deux octets

- Le premier octet définit le code opératoire,
- le second définit le poids faible de l'adresse effective.
- Exemple : **LDA \$08** ou **LDA < \$08** → Charge l'Accumulateur A avec le contenu dont l'adresse est formée par [DP] et l'opérande
→ chargement de l'accumulateur A avec le contenu de \$ 2008 (DP= \$20)



3. Mode d'adressage direct

▪ Instructions sur trois octets

- Le code opératoire (sur 2 octet)
- L'opérande (1 octet) : Poids Faible (8 bits) de l'Adresse Effective

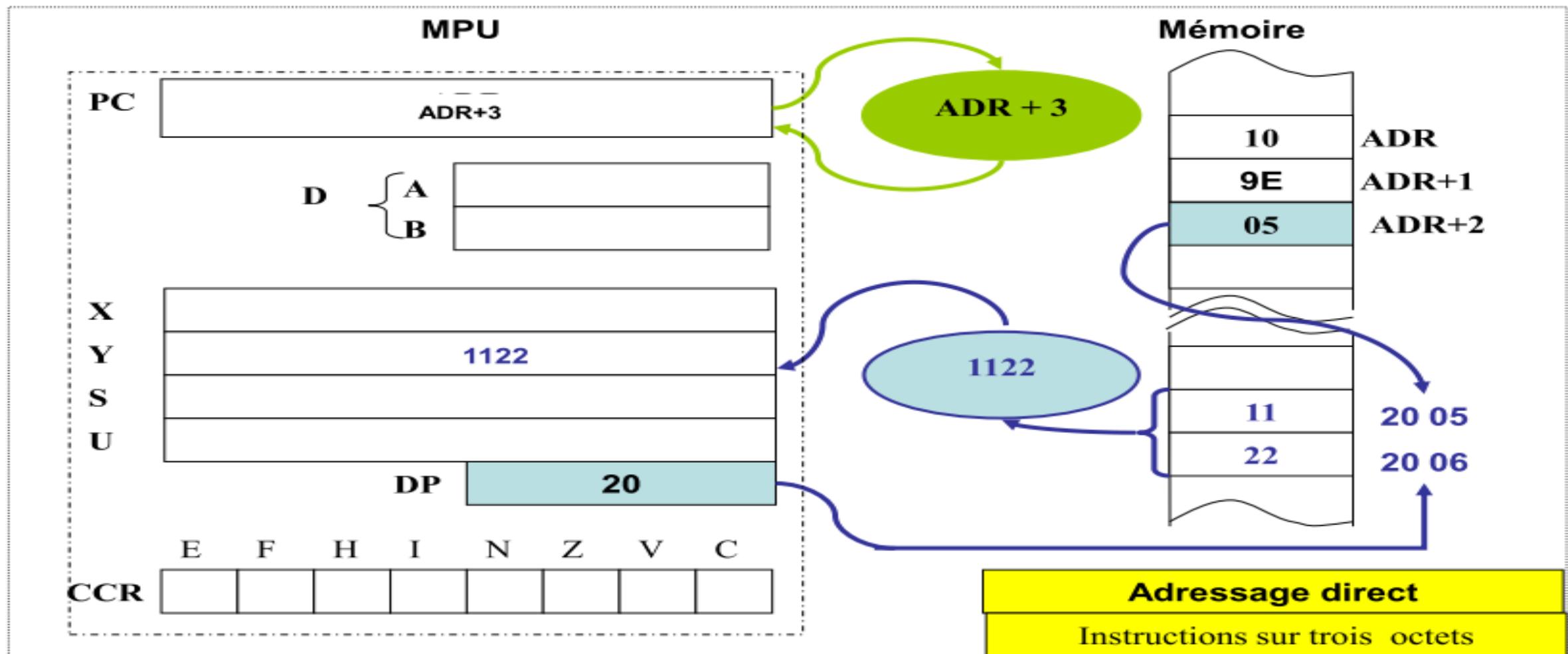
▪ Exemple : **LDY \$05** ou **LDY < \$05** → chargement du registre Y avec le contenu de \$2005/06 (DP=\$20)

→ Charge le registre Y avec le contenu sur 16 bits dont les adresses sont [DP] et partie basse et partie basse+1.

3. Mode d'adressage direct

▪ Instructions sur trois octets

- Exemple : LDY \$05 ou LDY < \$05 → chargement du registre Y avec le contenu de \$2005/06 (DP=\$20)



4. Mode d'adressage étendu (direct)

- ❖ Le code opératoire (sur 1 ou 2 octet)
- ❖ L'opérande (2 octet) : **Adresse Effective**

Le symbole « > » est une directive assembleur qui force l'adressage étendu.
Il existe deux types d'instructions dans ce mode d'adressage

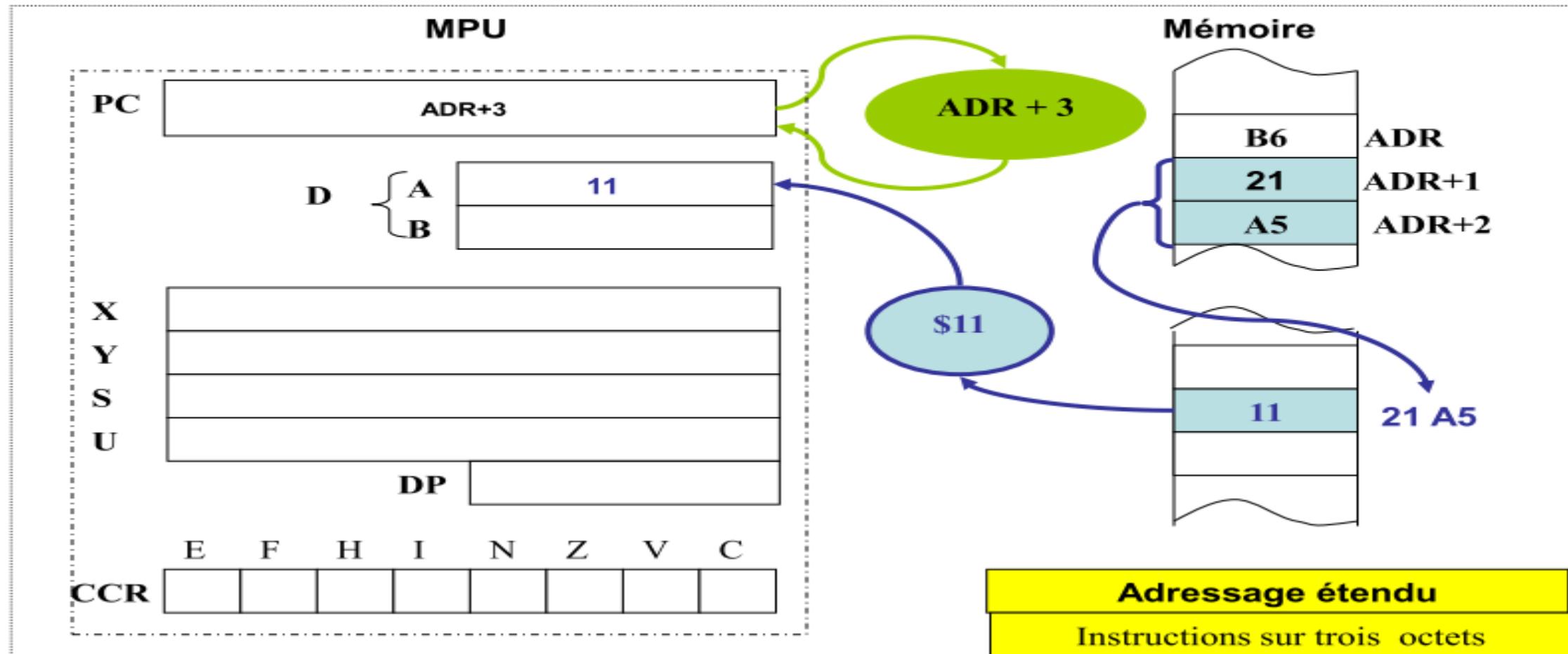
➤ Instructions sur trois octets

Le premier octet (code opératoire) est suivi de l'adresse 16 bits spécifiant l'emplacement de l'opérande (8 ou 16 bits).

- ❖ Le code opératoire (sur **1 octet**)
- ❖ L'opérande (**2 octet**) : **Adresse Effective**

4. Mode d'adressage étendu (direct)

Ex : LDA \$21A5 ou LDA > \$ 21A5 → chargement de l'accumulateur A avec le contenu de l'adresse \$21A5



4. Mode d'adressage étendu (direct)

➤ Instructions sur quatre octets

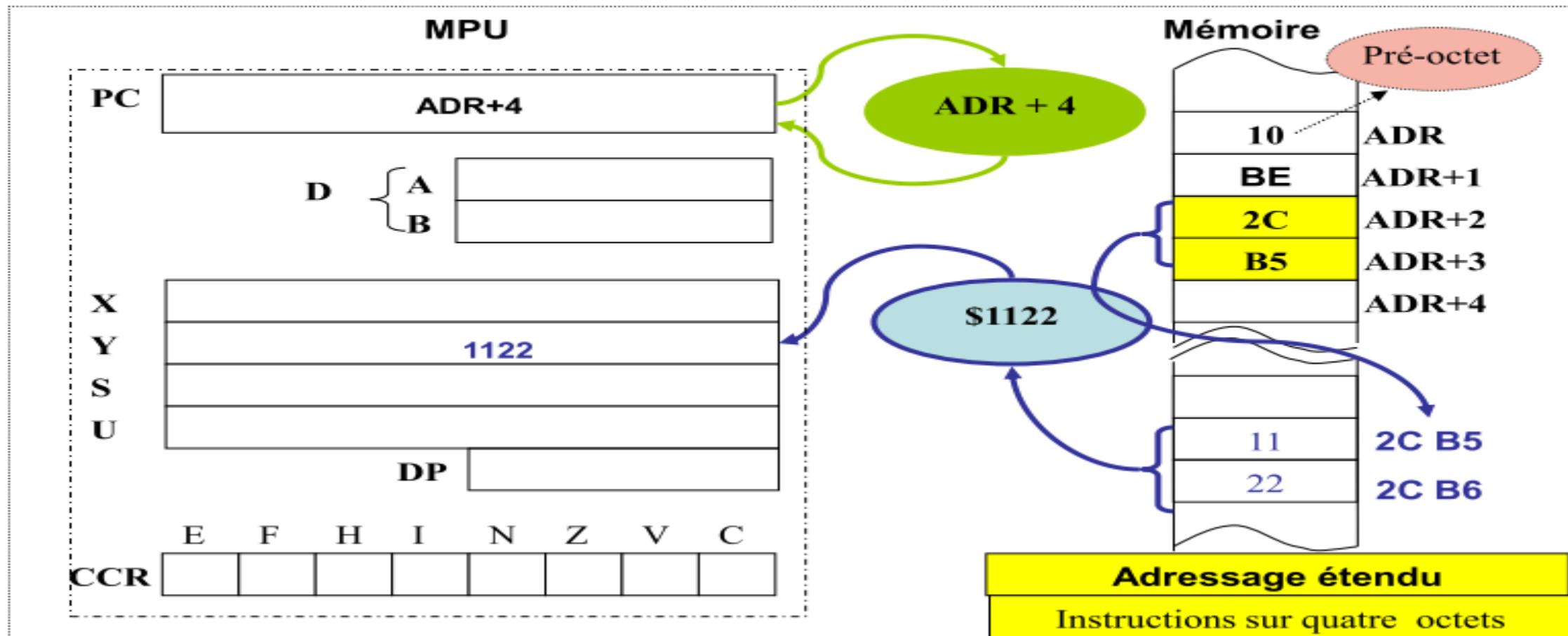
- ❖ Le code opératoire (sur **2 octet**)
- ❖ L'opérande (**2 octet**) : **A**dresse **E**ffective

Le premier octet est seulement nécessaire pour les instructions qui opèrent sur les pointeurs S et Y et sur les instructions de comparaison **CMPU, CMPD**.

Ex : **LDY \$2CB5** ou **LDY > \$2CB5** → chargement du registre Y Avec le contenu de \$2CB5

4. Mode d'adressage étendu (direct)

Ex :LDY \$2CB5 ou LDY > \$2CB5 → chargement du registre Y Avec le contenu de \$2CB5/B6



5. Mode d'adressage étendu indirect

Ce mode d'adressage est identique au mode d'adressage étendu mais il possède en plus une indirection.

La notation assembleur "[]" force l'adressage étendu indirect.

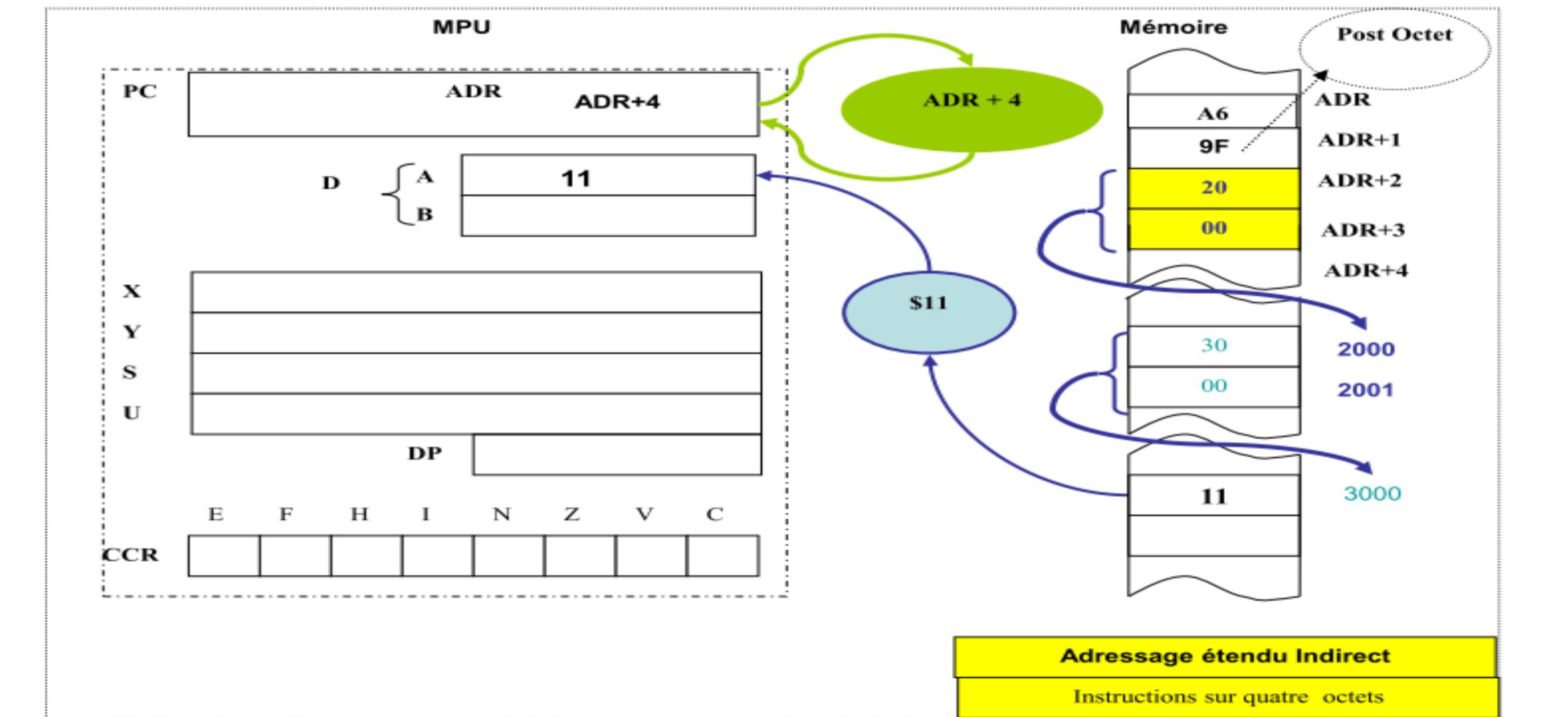
- ❖ Le code opératoire (sur **2 ou 3 octets**)
- ❖ L'opérande (**2 octet**) : **Adresse Effective**

➤ Instructions sur quatre octets

Les deux premiers octets déterminent le code opératoire ; code opératoire de l'adressage étendu simple suivi d'un **post-octet** déterminant **l'indirection**. Les 3ème et 4ème octets représentent l'adresse de transit.

5. Mode d'adressage étendu indirect

Ex : **LDA [\$2000]** → chargement de l'accumulateur avec le contenu dont l'adresse se trouve en \$2000 et \$2001.



Rq: Le post-octet est toujours \$9F.

5. Mode d'adressage étendu indirect

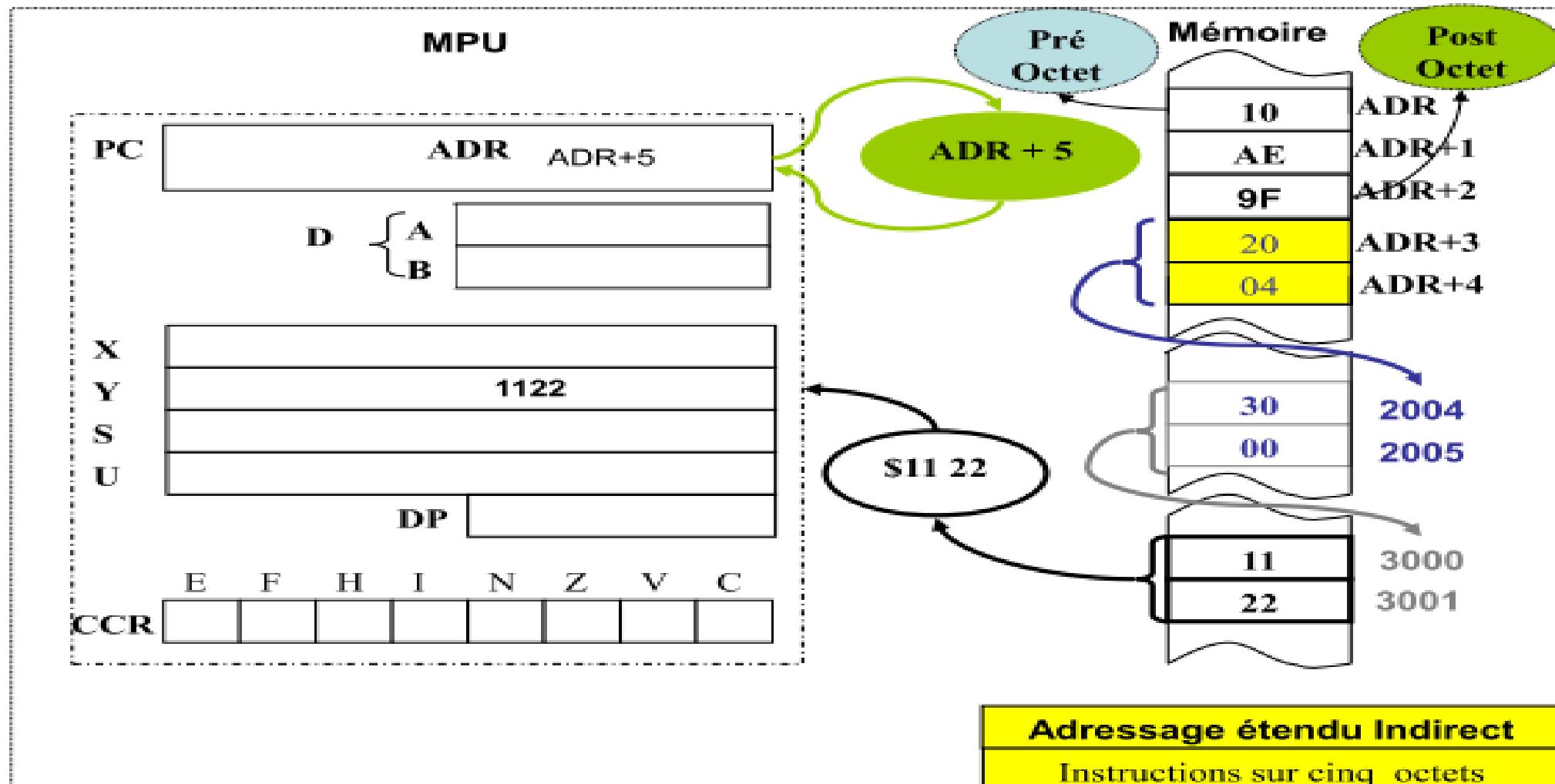
➤ Instructions sur cinq octets

Pour certaines instructions, il est nécessaire d'ajouter un pré-octet. Cela est nécessaire pour les instructions opérant sur les pointeurs **S** et **Y** (\$10) et pour les instructions de comparaison **CMPU** et **CMPD** (\$11). Le reste de la codification est identique au cas précédent.

Ex:LDY [\$2004] → chargement du registre d'index Y avec le contenu dont l'adresse se trouve en \$2004 et \$2005.

5. Mode d'adressage étendu indirect

Ex:LDY [\$2004] → chargement du registre d'index Y avec le contenu dont l'adresse se trouve en \$2004 et \$2005.

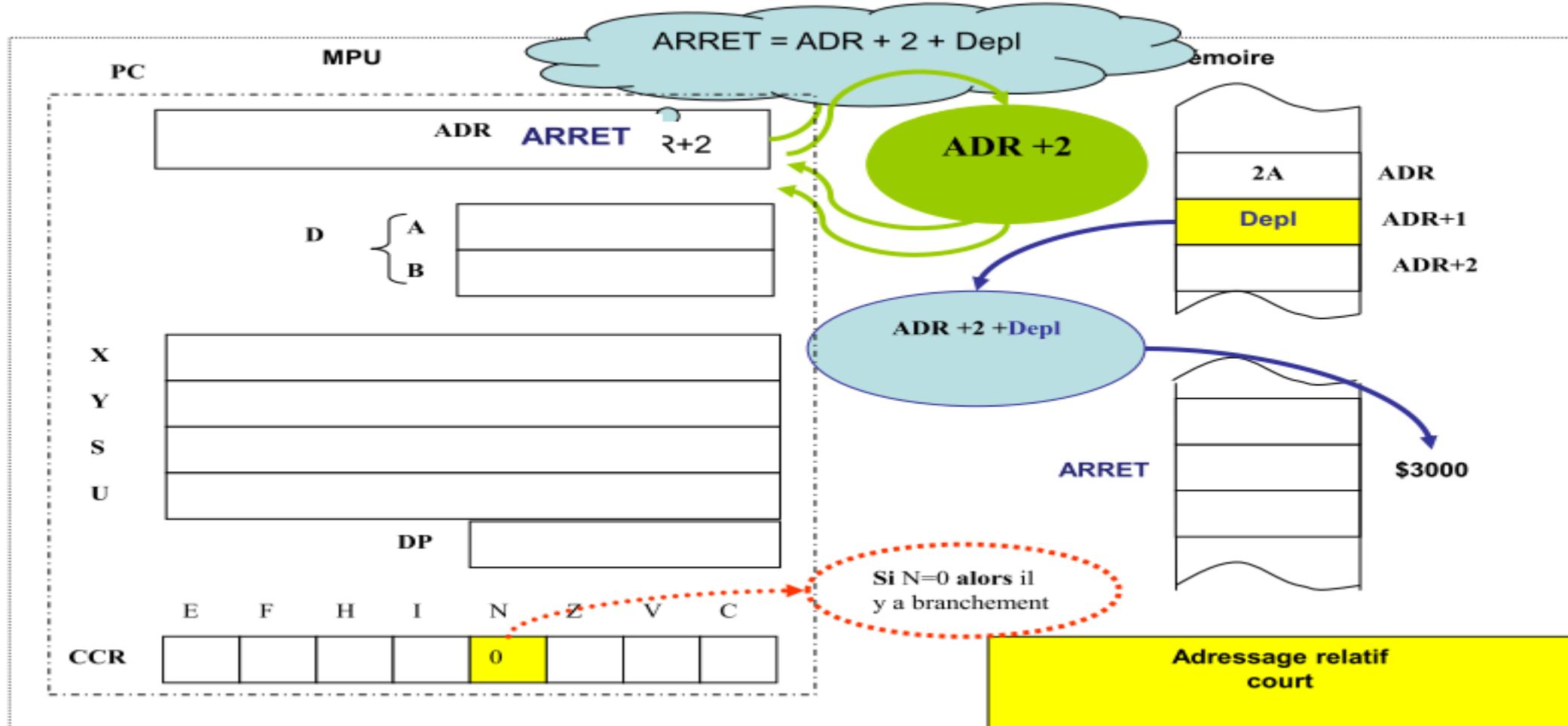


6. Mode d'adressage relatif court

- Ce mode d'adressage est réservé pour **les instructions de branchement**
 - L'instruction de branchement opérant en adressage relatif court fait appel à un format de **2 Octets** seulement.
 - **Le premier octet** détermine le OP Code qui spécifie le type de branchement en même temps que le test correspondant. **Le second octet** est la valeur du déplacement qui peut être positif ou négatif
 - En mode d'adressage relatif court, le déplacement étant codé sur **un octet**.
-
- Exemple:
Arret EQU \$3000
BPL Arret
 - Dans ce cas, BPL (Branch if Plus) fait un test sur le bit N du registre CCR , le branchement aura lieu si N=0 (Résultat de l'opération précédente positif)

6. Mode d'adressage relatif court

Ex: Arret EQU \$3000
BPL ARRET



Le déplacement (**Depl**) est calculé en fonction de la valeur de **ADR** par rapport à l'étiquette **ARRET** (\$3000).

6. Mode d'adressage relatif court

```
ADDA    $53F8      ; addition de la case mémoire $53F8 avec A
BEQ     FIN        ; si le contenu de A=0 après l'addition alors
.          ; branchement vers FIN (bit Z de CC est à 1)
.          ; si non on poursuit après l'instruction BEQ
FIN     SWI        ;
```

7. Mode d'adressage relatif long

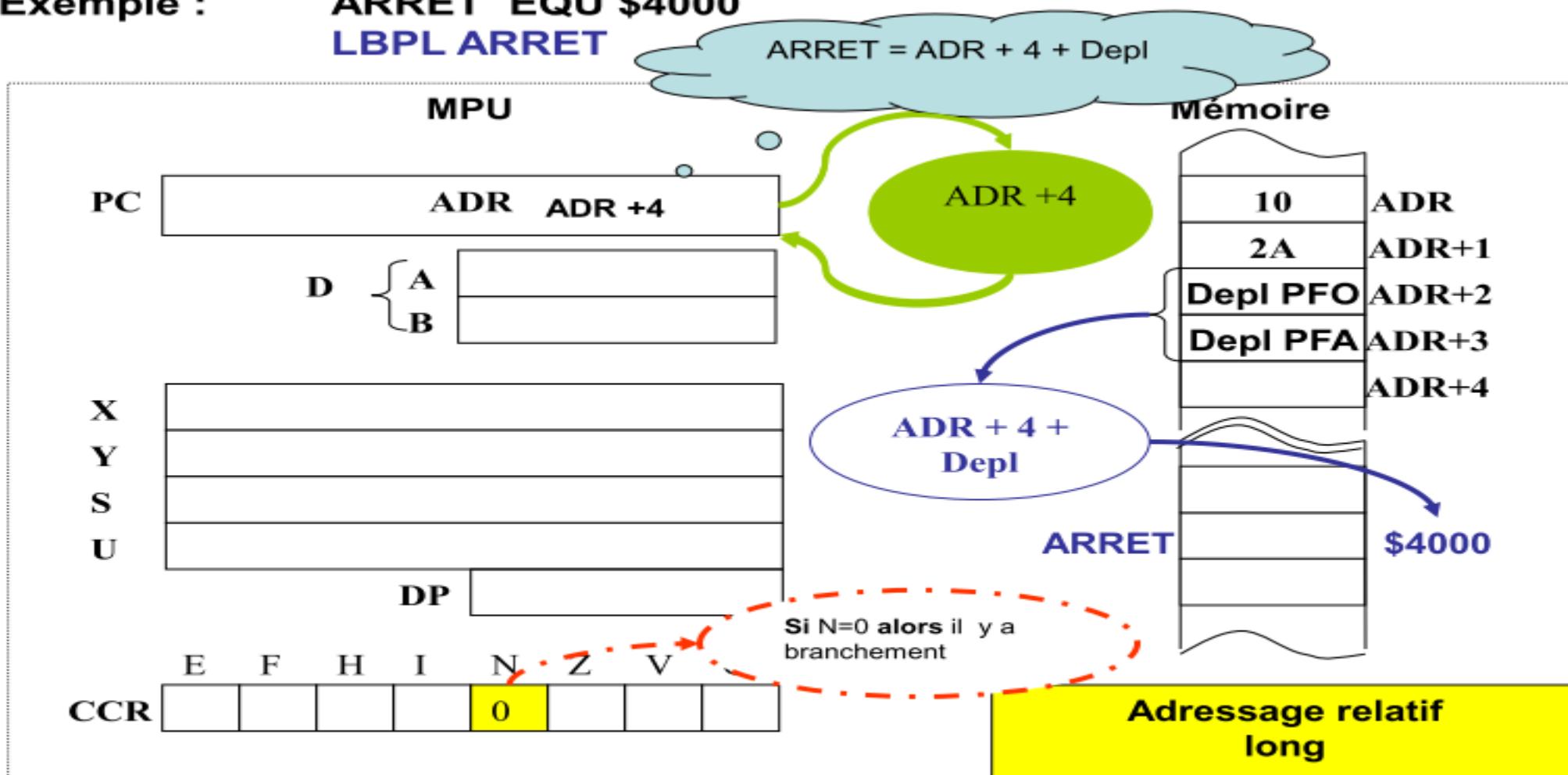
- Ce mode est identique au précédent, il est toujours réservé aux **branchements**.
- Les instructions sont codées sur quatre octets, les deux premiers déterminent le code opération, les 3ème et 4ème octets donnent la valeur signée du déplacement.
- Le déplacement est codé sur 16 bits.
 - Exemple :

ARRET EQU \$4000
LBPL ARRET

7. Mode d'adressage relatif long

Exemple :

ARRET EQU \$4000
LBPL ARRET



La présence d'un pré-octet (\$10) permet de différencier l'adressage relatif long de l'adressage relatif court.

8. Mode d'adressage indexé

Le mode d'adressage indexé présente l'avantage de pouvoir travailler en indirection. Dans ce cas on accède à l'adresse effective en transitant par une adresse intermédiaire

Registre

$$\textbf{ADRESSE EFFECTIVE} = \text{BASE} + \text{DEPLACEMENT}$$

La puissance d'un mode d'adressage indexé est déterminée par l'éventail des bases dont on dispose et par toutes les possibilités que l'on a.

la base peut être soit

- un des deux registres d'index (**X** ou **Y**) (ce qui est normale),
- mais aussi un des deux pointeurs de pile (**U** ou **S**)
- Ou ce qui est très intéressant, le compteur programme **PC** lui-même (l'adressage est alors, un cas particulier de l'adressage relatif).

Mode d'adressage indexé

L'offset, s'il existe, indique la distance relative en octets entre l'adresse effective et l'adresse de base.

$$\text{EA} = \text{Offset} + \text{Base}$$

Effective
Adresse

Base peut être
X, Y, S, U ou PC

Déplacement peut être :

- Constant (**nul, 5, 8 ou 16 bits**)
- Variable utilisation de **A, B ou D**
- Auto Incrémenté sur X, Y U ou S
- Auto Décrémenté sur X, Y U ou S
- Depuis PC en 8 bits
- Depuis PC en 16 bits

Mode d'adressage indexé

Pour ce qui est du **déplacement**, il y a de multiples possibilités: celui-ci peut être **nul**, codé sur **cinq**, **huit** ou **seize** bits, ou **variable** dans le cas de l'utilisation d'un accumulateur A, B ou D.

LDB \$20, X

Instruction déplacement Base : X ou Y ou U ou S ou PC

Enfin, l'adressage indexé offre des possibilités d'auto-incrémantation ou décrémantation de 1 ou de 2.

LDA , Y+

Toutes ces options sont sélectionnées par le **post-octet** qui suit le code opératoire.

Mode d'adressage indexé

Bit du registre post octet								Mode d'adressage indexé
7	6	5	4	3	2	1	0	AE= , Base + Déplacement
0	R	Déplacement						AE= , R ± 4 bits
1	R	0	0	0	0	0	0	AE= , R +
1	R	0/1	0	0	0	0	1	AE= , R ++
1	R	0	0	0	1	0	0	AE= , - R
1	R	0/1	0	0	1	1	0	AE= , - -R
1	R	0/1	0	1	0	0	0	AE= , R ± 0
1	R	0/1	0	1	0	0	1	AE= , R ± Acc B
1	R	0/1	0	1	1	0	0	AE= , R ± Acc A
1	R	0/1	1	0	0	0	0	AE= , R ± 7 bits
1	R	0/1	1	0	0	0	1	AE= , R ± 15 bits
1	R	0/1	1	0	1	1	1	AE= , R ± D (ACC A +Acc B)
1	0/1		1	1	0	0	0	AE= , PC ± 7 bits
1	0/1		1	1	0	0	1	AE= , PC ± 15 bits

→ Déplacement const
 → Auto Increm/Décrem
 → Déplacement nul
 → Dépl accumul
 → Déplacement const
 → Dépl accumul
 → Dépl PC

Mode d'adressage indexé

Les bits 5 et 6 du post octet permettent de définir la base :

BASE R	b6	b5
Index X	0	0
Index Y	0	1
Pointeur U	1	0
Pointeur S	1	1
Compteur Programme	X	X

→ Indifférent, la sélection de la base PC se fait à l'aide des bits 2 et 3 (1,1)

le **bit 7** définit le rôle du bit4 :

- b7=0 → b4 = bit de signe.
- b7=1 → b4= choix du mode direct (b4=0) ou indirect (b4=1);

Les bits 0 à 3 définissent le **champ du mode d'adressage**.

Nous allons voir toutes les combinaisons possibles.

Mode indexé. Déplacement nul.

Dans ce mode, le registre pointeur sélectionné contient l'adresse effective des données devant être utilisées par l'instruction. Ce mode est le mode indexé le plus rapide. Il existe deux types d'instructions :

➤ Instructions sur deux octets :

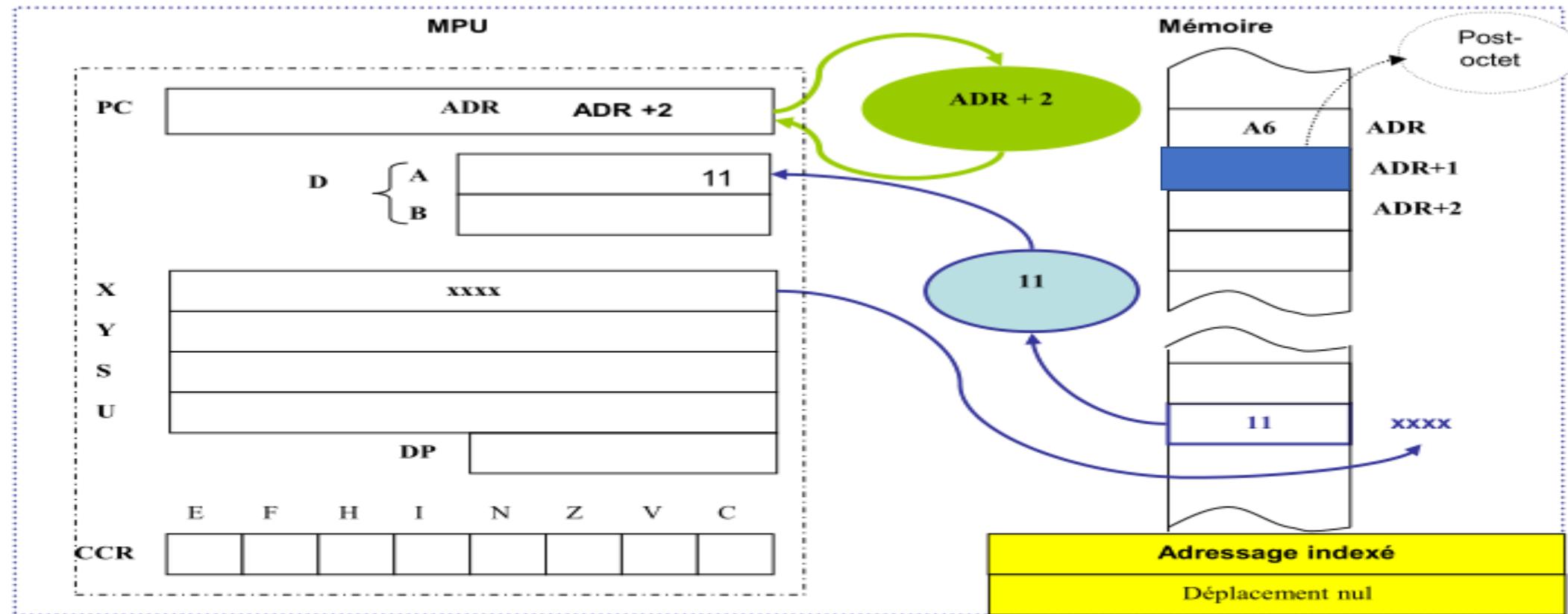
Le code opératoire est suivi du **post-octet** précisant les options choisies pour l'instruction en cours.

Exemple :

LDA ,X → chargement de A avec la valeur dont l'adresse est le contenu de l'index X.

Mode indexé. Déplacement nul.

LDA ,X → chargement de A avec la valeur dont l'adresse est le contenu de l'index X.



Dans ce cas le post-octet est égale à \$84 :

- b7=1 → b4=0 signifie que l'adressage est indexé direct
- b6.b5=0.0 → l'index est X
- b3.b2.b1.b0=0.1.0.0 → le déplacement est nul.



Mode d'adressage indexé. Auto incrémentation/décrémentation

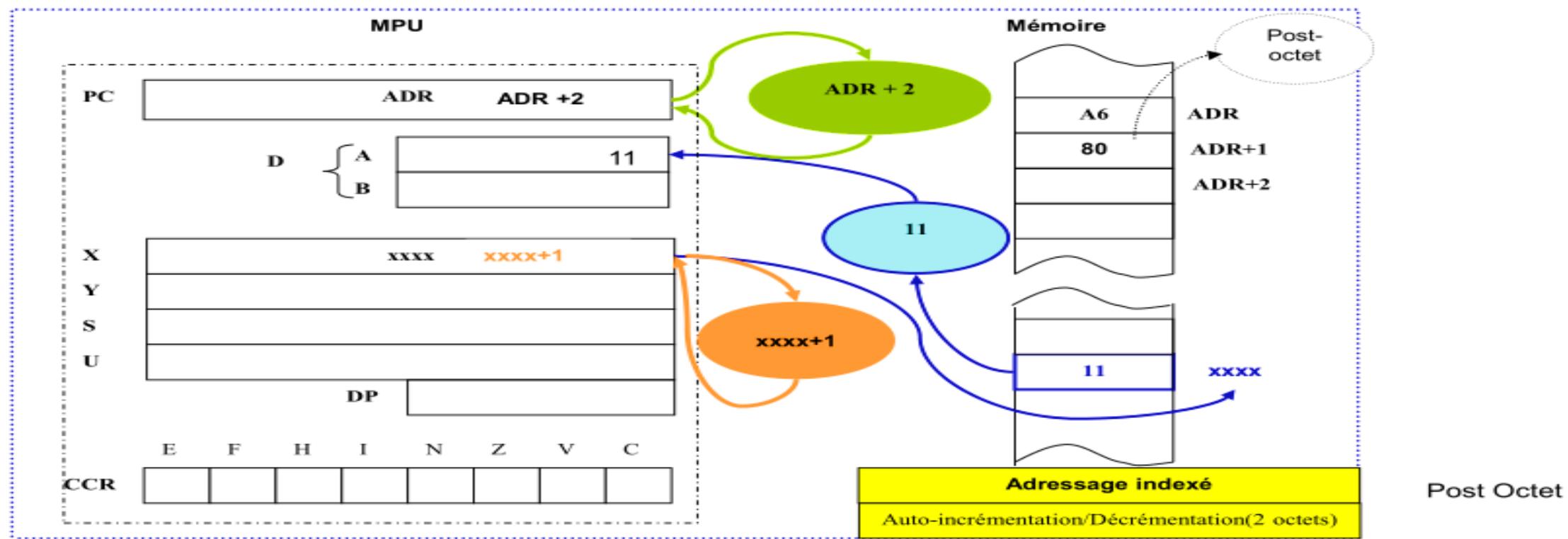
- On peut obtenir des pré-décrémentations de un ou deux et des post-incrémentations de un ou deux.
- Ces possibilités permettent de gérer facilement une table.
- Le registre pointeur sélectionné contient l'adresse effective des données utilisées par l'instruction.

➤ Instructions sur deux octets

- Exemple : LDA ,X+
→ chargement de A avec la valeur dont l'adresse est le contenu de X, post-incrémantation par un de X.

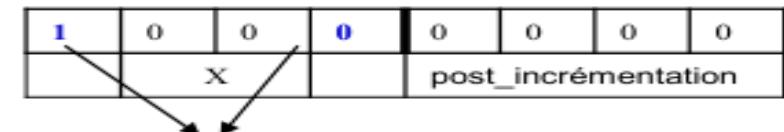
Mode d'adressage indexé. Auto incrémentation/décrémentation

LDA ,X+ → chargement de A avec la valeur dont l'adresse est le contenu de l'index X.post incrémentation par un de X



Dans ce cas le post-octet est égale \$80 :

- b7=1 → b4=0 signifie que l'adressage est indexé direct
- b6.b5=0.0 → l'index est X
- b3.b2.b1.b0=0.0.0.0 → mode post_incrémantion.



Adressage indexé direct

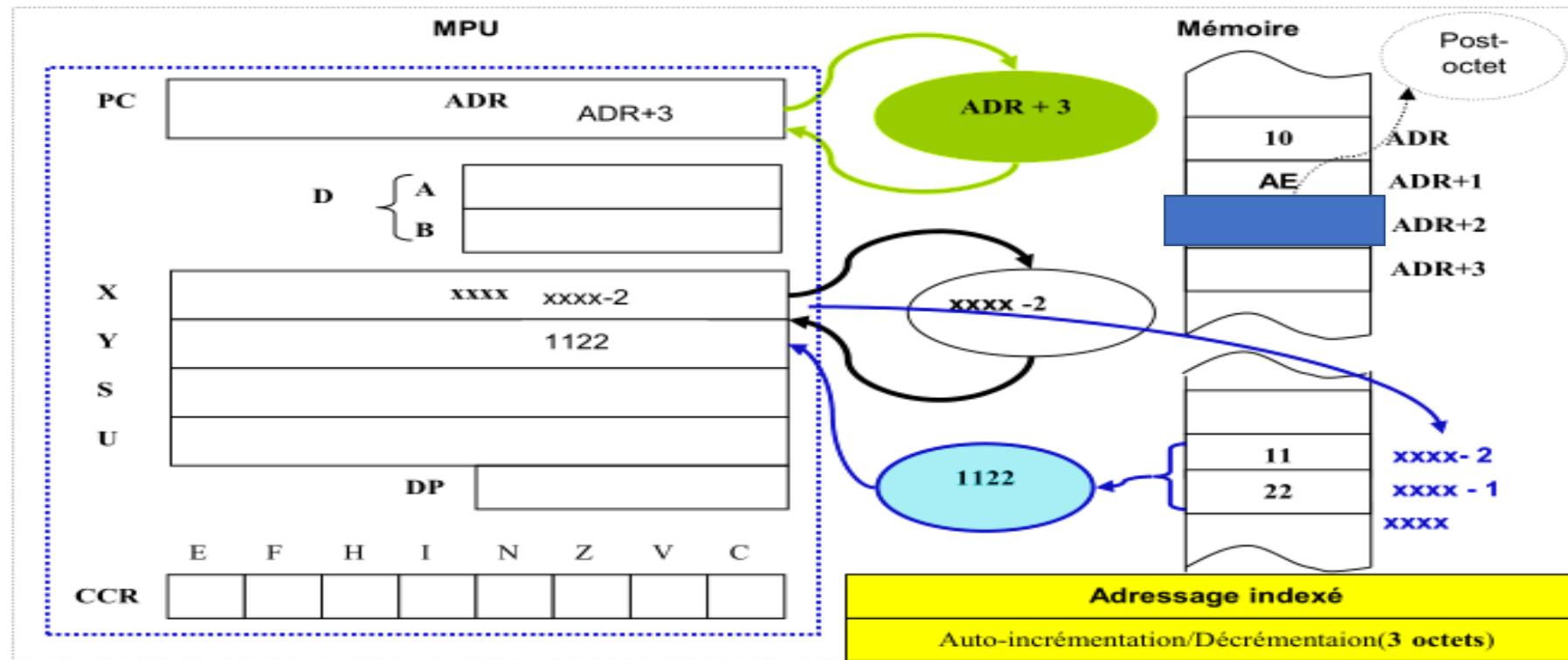
NB: Le mode auto-incrémantion/décrémantion par un est utilisé pour gérer des tables de données

1	R	0	0	0	0	0	AE= , R +
1	R	0/1	0	0	0	1	AE= , R ++
1	R	0	0	0	1	0	AE= , - R
1	R	0/1	0	0	1	1	AE= , - - R

→ Auto Increm/Décrem

Mode d'adressage indexé. Auto incrémentation/décrémentation

LDY , --X → chargement de Y avec la valeur dont l'adresse de base est le contenu de X - 2.



Dans ce cas le post-octet est égale \$83 :

- b7=1 → b4=0 signifie que l'adressage est indexé direct
- b6.b5=0.0 → l'index est X
- b3.b2.b1.b0=0.0.1.1 → double pré décrémentation.

NB: Le mode auto-incrémentation/décrémentation par deux est utilisé pour gérer des **tables d'adresses**

1	R	0	0	0	0	0	AE= , R +
1	R	0/1	0	0	0	1	AE= , R ++
1	R	0	0	0	1	0	AE= , - R
1	R	0/1	0	0	1	1	AE= , - -R

Auto Increm/Décrem

Mode d'adressage indexé. Déplacement constant 4 bits

Dans ce mode d'adressage, l'adresse effective de l'opérande est la somme du déplacement (en complément à deux) et du contenu du registre constituant la base.

Le registre de base n'est pas modifié. Il existe trois formes d'adressage indexé à déplacement constant, suivant la valeur de cette constante.

➤ Déplacement sur ± 4 bits

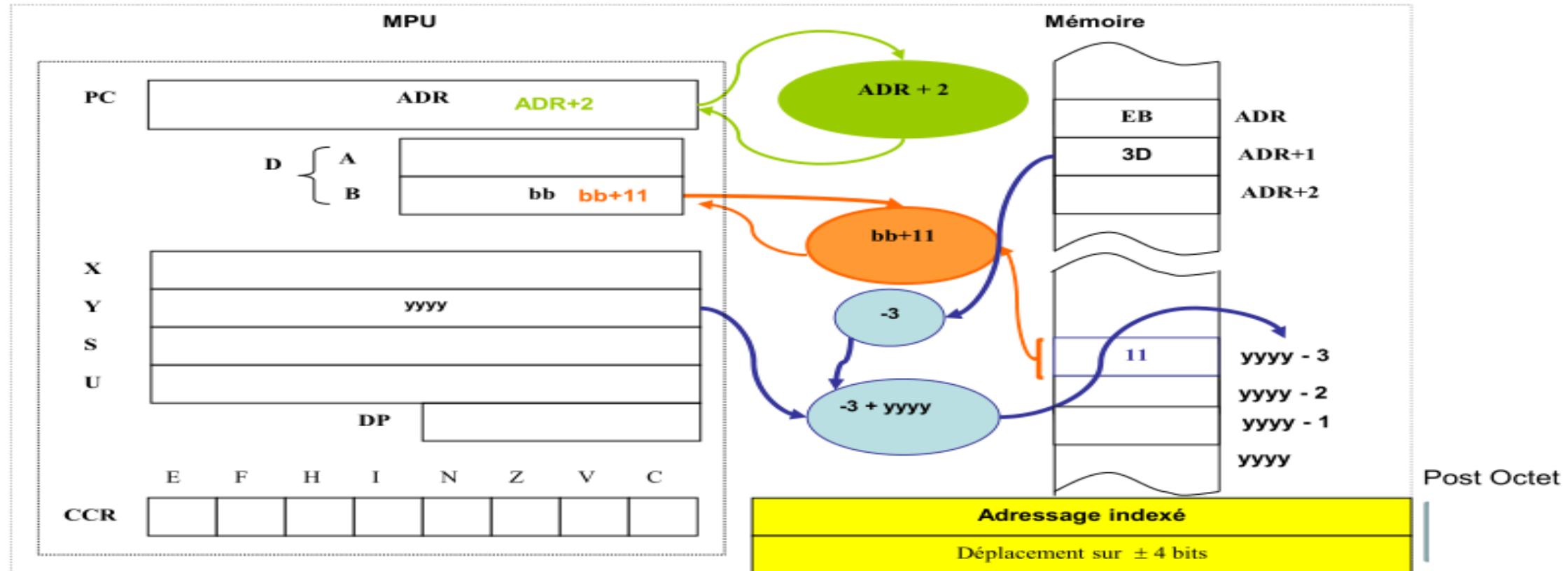
Ce déplacement codé sur 5 bits (en complément à deux) présente l'avantage d'être contenu dans le post-octet d'indexation. Ce qui permet un gain de place mémoire et une exécution plus rapide de cette instruction.

Les bits 0 à 4 du post-octet déterminent donc la valeur du déplacement qui peut être de -16 octets en arrière (1 0000) et de + 15 octets en avant

(0 1111). Dans ce cas le bit constamment à zéro initialise le bit 4 comme bit de signe.

Mode d'adressage indexé. Déplacement constant 4 bits

ADDB -3,Y → addition du contenu d'adresse mémoire $Y-3$ au contenu de l'accumulateur B, le résultat est dans B.



Le post-octet prend la valeur \$3D :

- b7=0 → le bit b4 est le bit de signe de déplacement.
- b6.b5=0.1 → index Y .
- b4.b3.b2.b1.b0=1.1.1.0.1 → le déplacement est -3 en complément à deux.

0	0	1	1	1	1	0	1
Y						Déplacement (en compl à 2) = -3	

Bit du registre post octet								Mode d'adressage indexé			
7	6	5	4	3	2	1	0	AE= , Base + Déplacement			
0	R	Déplacement						AE= , R \pm 4 bits			

→ Déplacement const

Mode d'adressage indexé. Déplacement constant 7 bits

➤ Déplacement sur \pm 7 bits

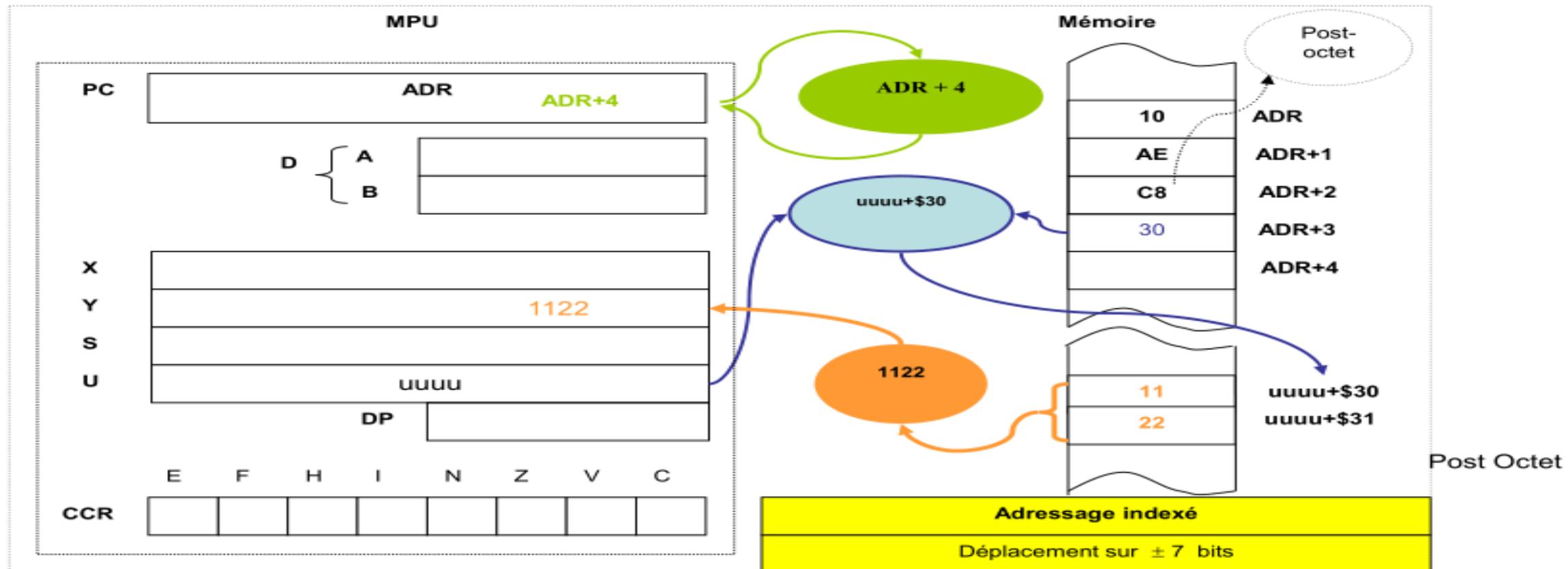
Ce déplacement codé sur 8 bits (en complément à deux) est contenu dans un seul octet, placé à la suite du code opératoire proprement dit et du post-octet. Les déplacements possibles sont donc compris entre – 128 et + 127 octets.

Exemple :

LDY \$30, U → chargement du pointeur Y avec le contenu mémoire dont l'adresse de base est le contenu de U + \$30

Mode d'adressage indexé. Déplacement constant 7 bits

LDY \$30, U → chargement du pointeur Y avec le contenu mémoire dont l'adresse de base est le contenu de U + \$30



Le post-octet prend la valeur **C8** en hexa (1100 1000) :

- b7=1 → b4=0 → adressage indexé direct.
- b6.b5=1.0. → index U .
- b3.b2.b1.b0=1.0.0.0 → le déplacement est codé sur 8 bits en compléments à deux.



Adressage indexé direct

1	R	0/1	1	0	0	0	AE= , R ± 7 bits
1	R	0/1	1	0	0	1	AE= , R ± 15 bits

→ Déplacement const

Mode d'adressage indexé. Déplacement constant 15bits

➤ Déplacement sur ± 15 bits

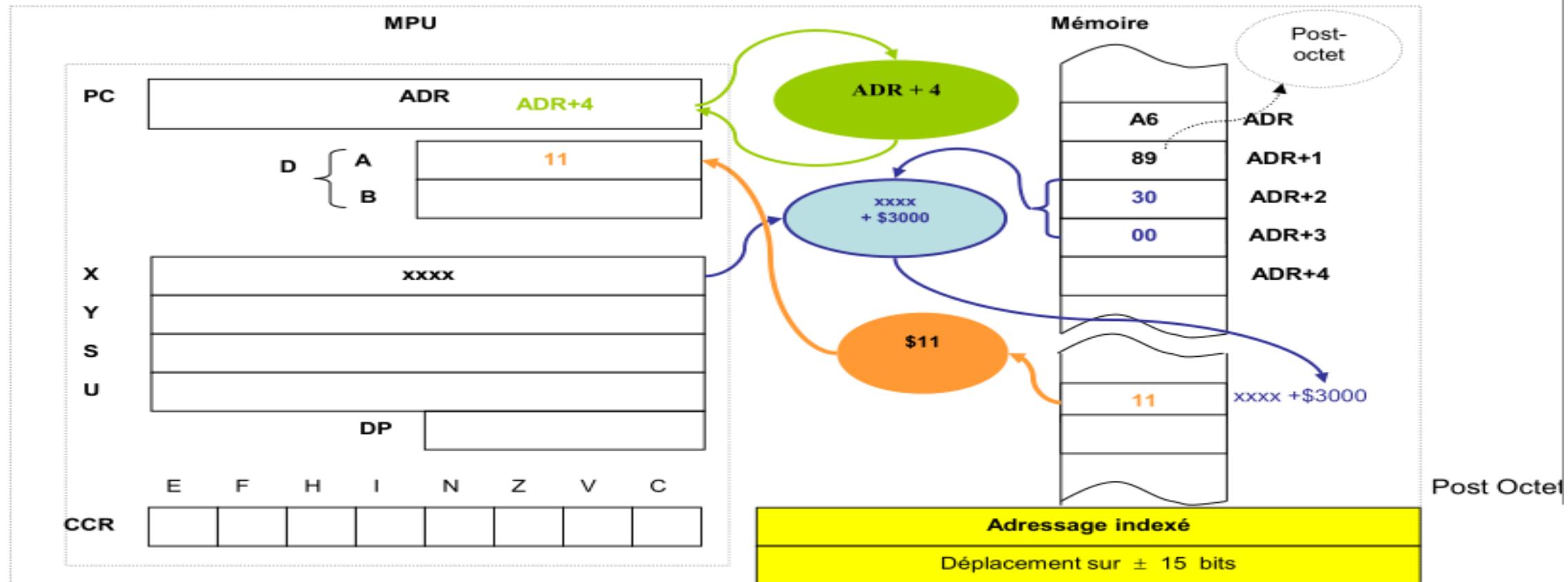
Ce déplacement codé sur 16bits (en complément à deux) est contenu dans deux octets placés à la suite de ceux de l'instruction (code opératoire + post octet). Les déplacements possibles sont donc compris entre – 32768 et + 32767 octets.

Exemple :

LDA \$3000, X → chargement de l'accumulateur A avec le contenu mémoire d'adresse X + \$3000

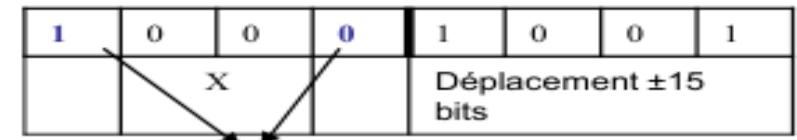
Mode d'adressage indexé. Déplacement constant 15bits

LDA \$3000, X → chargement de l'accumulateur A avec le contenu mémoire d'adresse X + \$3000



Le post-octet prend la valeur **89** en hexa (1000 1001) :

- b7=1 → b4=0 → adressage indexé direct.
- b6.b5=0.0. → index X .
- b3.b2.b1.b0=1.0.0.1 → le déplacement est codé sur 16 bits en compléments à deux.



Adressage indexé direct

1	R	0/1	1	0	0	0	AE= , R ± 7 bits
1	R	0/1	1	0	0	1	AE= , R ± 15 bits

→ Déplacement const

Mode d'adressage indexé. Déplacement accumulateur 7bits

Ce mode d'adressage est semblable au précédent à l'exception du déplacement qui n'est plus codé sur des octets spécifiques mais contenu dans les accumulateurs A,B ou D du microprocesseur..

L'adresse effective est donc la somme des registres pointeur et accumulateur spécifiés dans le mnémonique de l'instruction

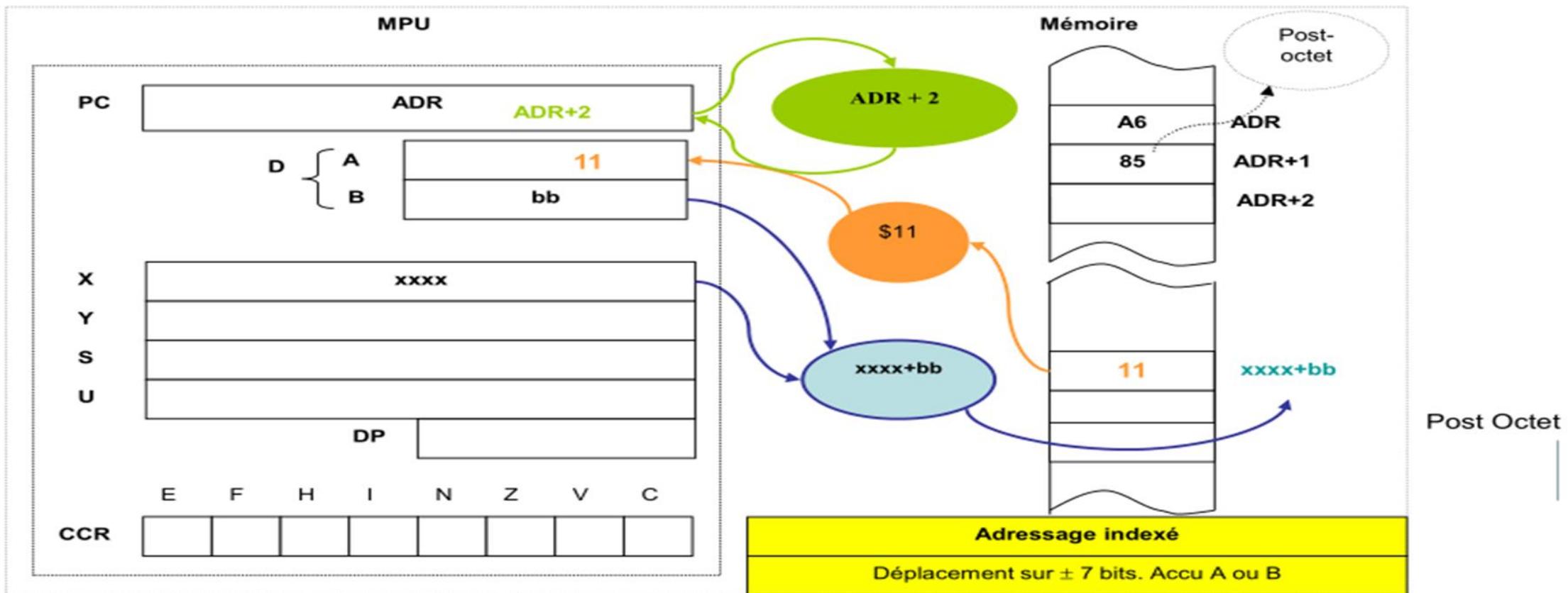
➤ Déplacement sur ± 7 bits. Accumulateurs A ou B

Exemple :

LDA B, X → chargement de l'accumulateur A avec le contenu mémoire d'adresse $X + B$

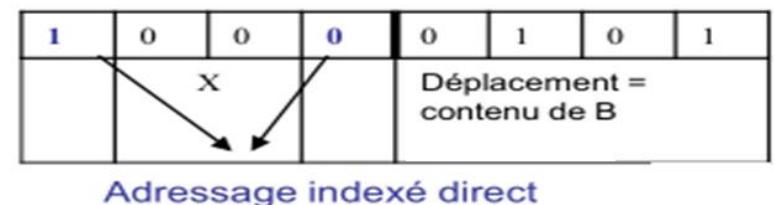
Mode d'adressage indexé. Déplacement accumulateur 7bits

LDA B, X → chargement de l'accumulateur A avec le contenu mémoire d'adresse X + B



Le post-octet prend la valeur **85** en hexa (1000 0101) :

- b7=1 → b4=0 → adressage indexé direct.
- b6.b5=0.0. → index X .
- b3.b2.b1.b0=0.1.0.1 → le déplacement est égale au contenu de l'Acc B

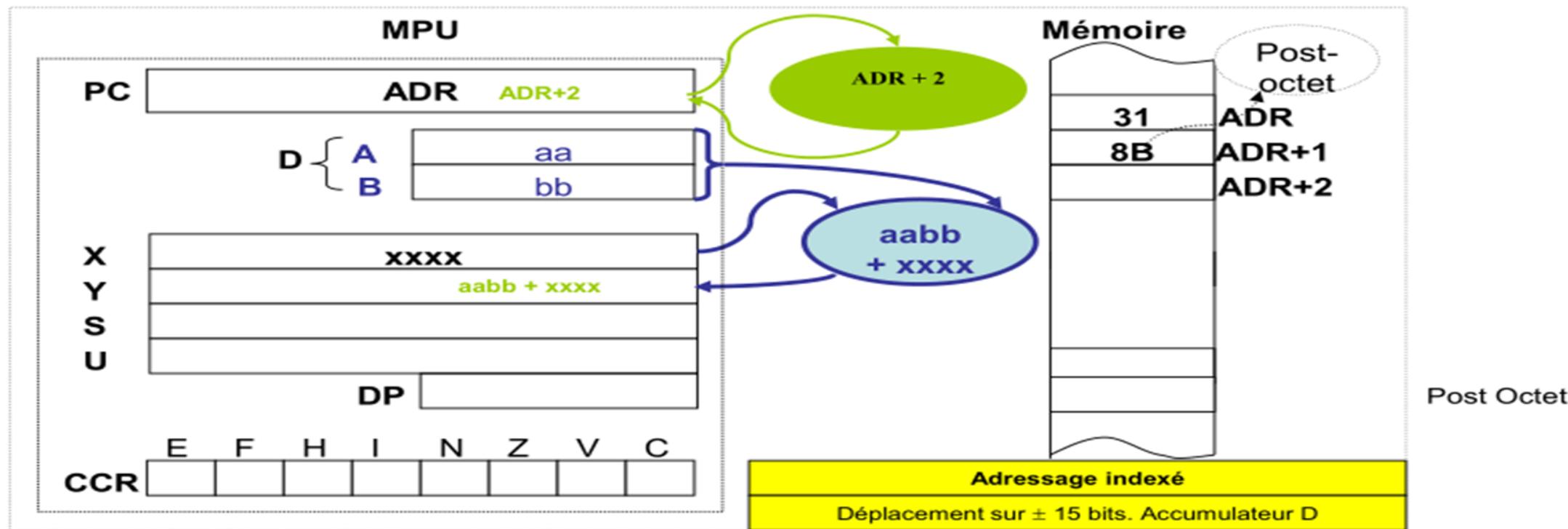


1	R	0/1	0	1	0	1		AE= , R ±Acc B
1	R	0/1	0	1	1	0		AE= , R ±Acc A

→ Dépl accumul

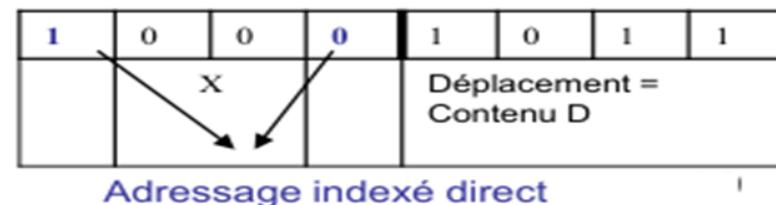
Mode d'adressage indexé. Déplacement accumulateur 15bits

LEAY D, X → chargement dans l'index Y de l'adresse effective donnée par la somme D + X



Le post-octet prend la valeur **8B** en hexa (1000 1011) :

- b7=1 → b4=0 → adressage indexé direct.
- b6.b5=0.0. → index X .
- b3.b2.b1.b0=1.0.1.1 → le déplacement = contenu de l'Acc D=(A :B)



1	R	0/1	1	0	1	1	AE= , R ± D (ACC A + ACC B)
---	---	-----	---	---	---	---	-----------------------------

→ Dépl accumul

Mode d'adressage indexé. Base=compteur programme « PC »

Les paragraphes précédents nous ont permis d'aborder les différents modes d'adressage indexé utilisant les pointeurs X, Y,S et U comme base.

L'utilisation du compteur ordinal comme base d'indexation impose des restrictions sur les types de déplacements.

Seuls les déplacements constants codés sur 8 ou 16 bits (en complément à 2) peuvent être utilisés.

Mode d'adressage indexé. Base=compteur programme « PC »

➤ Déplacement sur ± 7 bits.

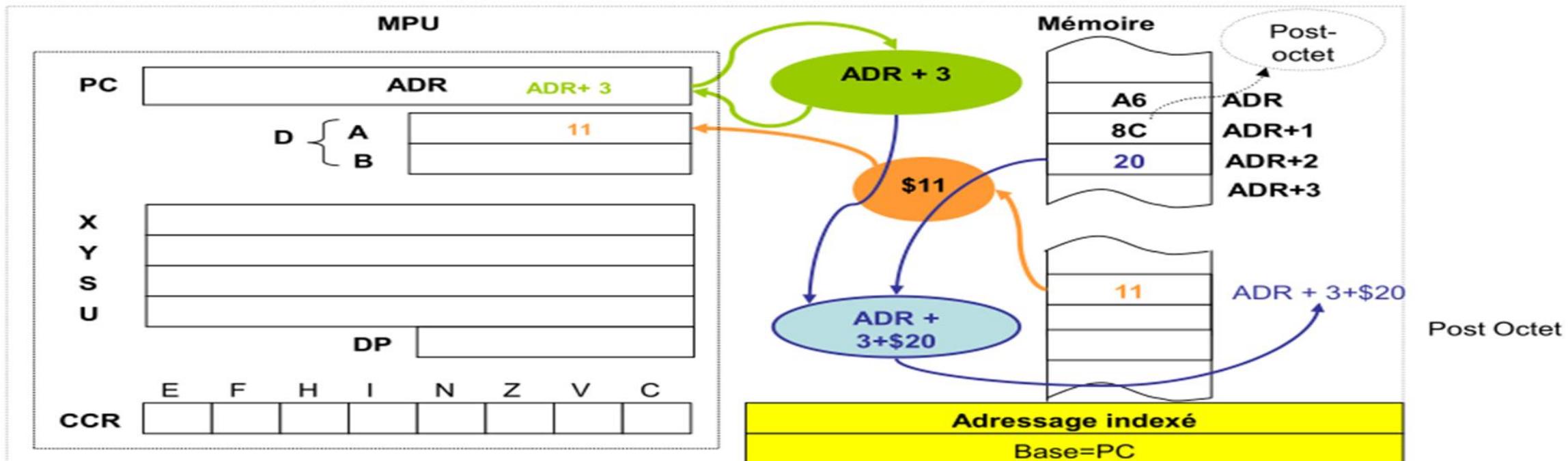
L'adresse effective est égale au compteur programme courant plus un déplacement codé sur 8 bits. L'instruction est codée sur 3 octets (code opératoire + post-octet+ déplacement).

Exemple :

LDA \$20, PC → chargement de l'accumulateur A avec le contenu mémoire dont d'adresse est la valeur de PC + \$20

Mode d'adressage indexé. Base=compteur programme 7 bits.

LDA \$20, PC → chargement de l'accumulateur A avec le contenu mémoire dont d'adresse est la valeur de PC + \$20



Le post-octet prend la valeur **8C** en hexa (1000 1100) :

- b7=1 → b4=0 → adressage indexé direct.
- b6.b5 → indifferent
- b3.b2.b1.b0=1.1.0.0 → le déplacement sur 8 bits

1	0	0	0	1	1	0	0
Indiférent				Déplacement sur 8 bits			

Adressage indexé direct

1	X	0/1	1	1	0	0	AE= , PC ± 7 bits
1	X	0/1	1	1	0	1	AE= , PC ± 15 bits

]} → Dépl PC

Mode d'adressage indexé. Base=compteur programme 15 bits.

➤ Déplacement sur ± 15 bits.

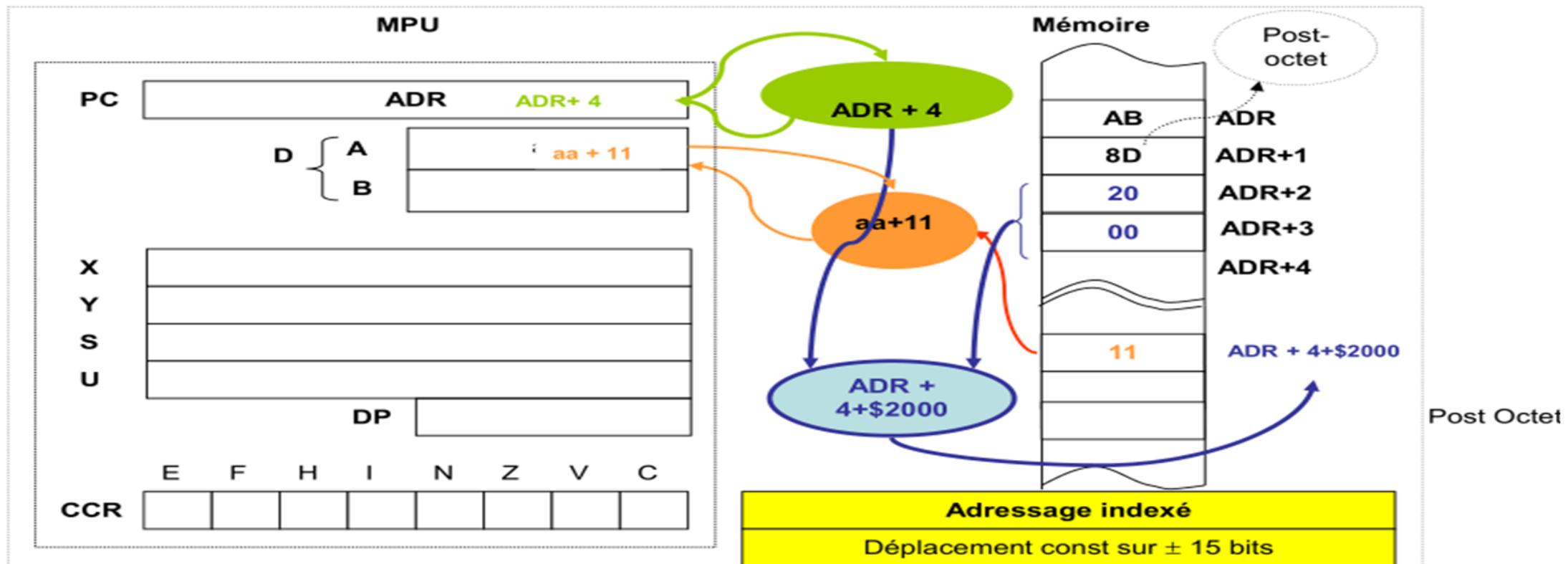
L'adresse effective est égale au compteur courant plus un déplacement codé sur deux octets. Ce mode d'adressage présente l'avantage de permettre des déplacements sur tout l'espace mémoire du processeur

Exemple :

ADDA \$2000, PC → addition du contenu mémoire dont l'adresse est la valeur courante du PC+ \$2000 et de l'accumulateur A, le résultat est dans A.

Mode d'adressage indexé. Base=compteur programme 15 bits.

ADDA \$2000, PC → addition du contenu mémoire dont l'adresse est la valeur courante du PC + \$2000 et de l'accumulateur A, le résultat est dans A.



Le post-octet prend la valeur **8D** en hexa (1000 1101) :

- b7=1 → b4=0 → adressage indexé direct.
- b6.b5 → indifferent
- b3.b2.b1.b0=1.1.0.1 → le déplacement sur 16 bits

1	0	0	0	1	1	0	1
Indiférent				Déplacement sur 16 bits			

Adressage indexé direct

1	X	0/1	1	1	0	0	AE= , PC ± 7 bits] → Dépl PC
1	X	0/1	1	1	0	1	AE= , PC ± 15 bits	

Mode d'adressage indexé: Étiquette localisée

➤ Étiquette localisée.

Dans ce cas, la syntaxe assembleur n'utilise pas directement un déplacement mais une **étiquette localisée** située à ± 128 octets de l'instruction suivante (déplacement sur 8 bits) ou n'importe où sur l'espace mémoire du microprocesseur (déplacement sur 16 bits).

Comme précédemment l'adresse effective est égale au compteur programme courant plus un déplacement codé sur 8 ou 16 bits. L'instruction est codée sur trois ou quatre octets.

La syntaxe assembleur contient le mnémonique de l'instruction, le nom de l'étiquette localisée plus la base.

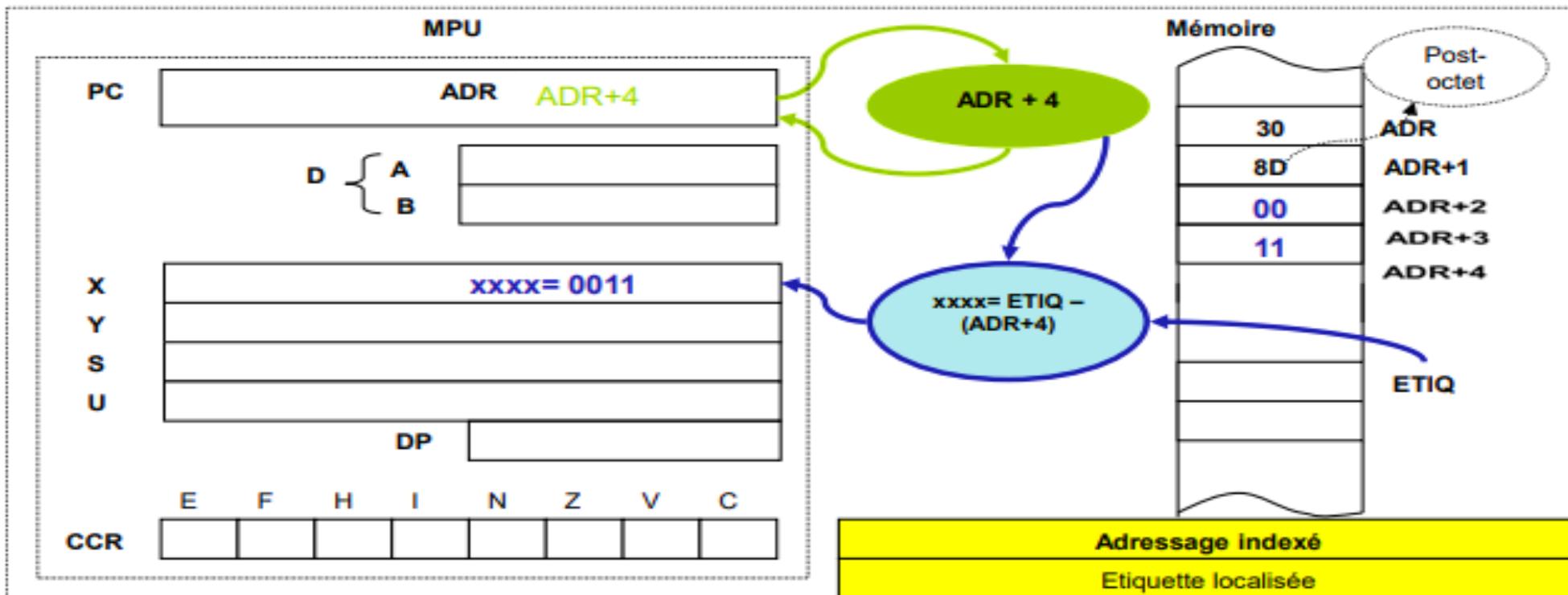
L'assembleur calcule la valeur du déplacement en fonction de la position de l'étiquette.

Mode d'adressage indexé: Étiquette localisée

LEAX ETIQ, PC → chargement du registre d'index X avec l'adresse effective qui est donnée par la position de l'étiquette par rapport au compteur programme courant.

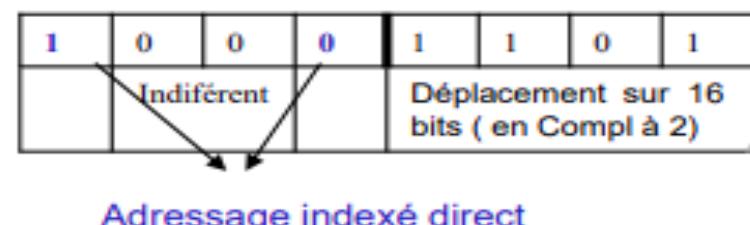
Dans cet exemple on prend :

$$\begin{aligned} \text{ADR} &= \$2000 & \rightarrow \text{xxxx} = 0011 \\ \text{ETIQ} &= \$2015 \end{aligned}$$



Le post-octet prend la valeur **8D** en hexa (1000 1101) :

- b7=1 → b4=0 → adressage indexé direct.
- b6.b5 → indifférent
- b3.b2.b1.b0=1.1.0.1 → le déplacement sur 16 bits (en complément à 2)



8. Mode d'adressage indexé

Mode adressage	Exemple
Mode adressage indexé	<p>1. Déplacement NUL :</p> <ul style="list-style-type: none">a. Sur deux octetsa. Sur trois octets <p>2. Auto Incrémentation/décrémentation :</p> <ul style="list-style-type: none">a. Sur deux octetsa. Sur trois octets <p>3. Déplacement constant :</p> <ul style="list-style-type: none">a. Sur ± 4 bitsa. Sur ± 7 bitsa. Sur ± 15 bits <p>4 .Déplacement accumulateur :</p> <ul style="list-style-type: none">a. Déplacement sur ± 7 bits. Acc A ou Ba. Déplacement sur ± 15 bits. Acc D <p>5. Base=Compteur Programme PC</p> <ul style="list-style-type: none">a. Déplacement sur ± 7 bits.a. Déplacement sur ± 15 bits.a. Etiquette localisée
	LDB ,X
	LDY , U
	LDA ,Y+
	LDY ,--X
	ADDB -14,X
	LDA \$30,X
	LDB \$8000,Y
	LDA B,X
	LEAY D,X
	LDB \$20,PC
	ADDA \$2000,PC
	LEAX ETIQ,PC

9. Mode d'adressage indexé indirect

Le mode d'adressage indexé présente l'avantage de pouvoir travailler en indirection.

Dans ce cas on accède à l'adresse effective en transitant par une adresse intermédiaire.

On peut donc écrire :

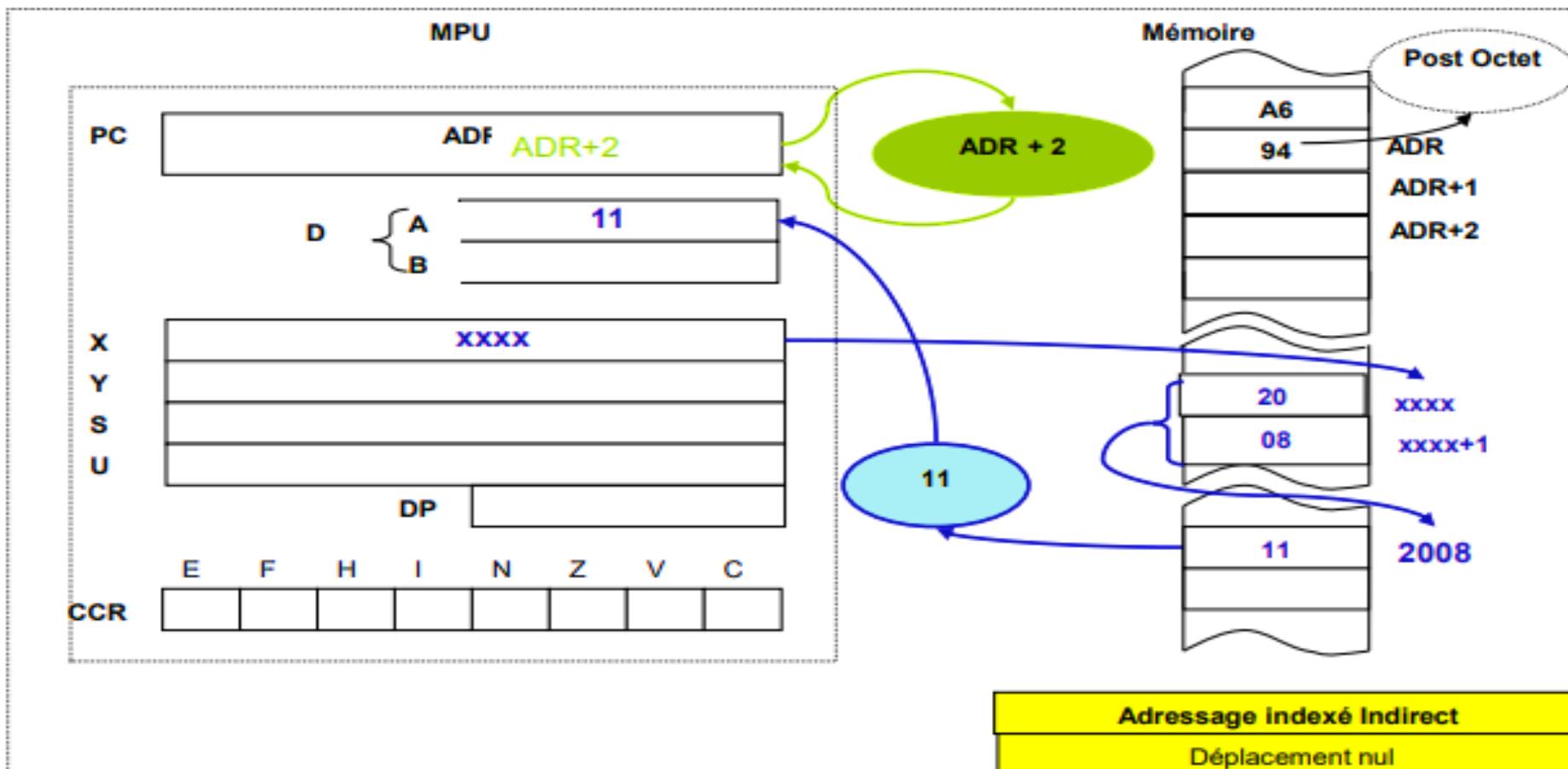
Adresse effective = contenu mémoire dont l'adresse de base est la somme de l'index + le déplacement.

La syntaxe assembleur utilisée pour définir l'indirection est la même que celle de l'adressage étendu : « [] »

Les instructions sont codés sur deux octets, le premier définit le code opératoire (identique à l'indexé simple), le second le type de déplacement, le type de base et l'indirection.

Mode d'adressage indexé indirect. Déplacement nul

LDA [, X] → chargement de A avec le contenu mémoire dont l'adresse est le contenu de (X,X+1).



Le post-octet prend la valeur 94 en hexa (1001 0100) :

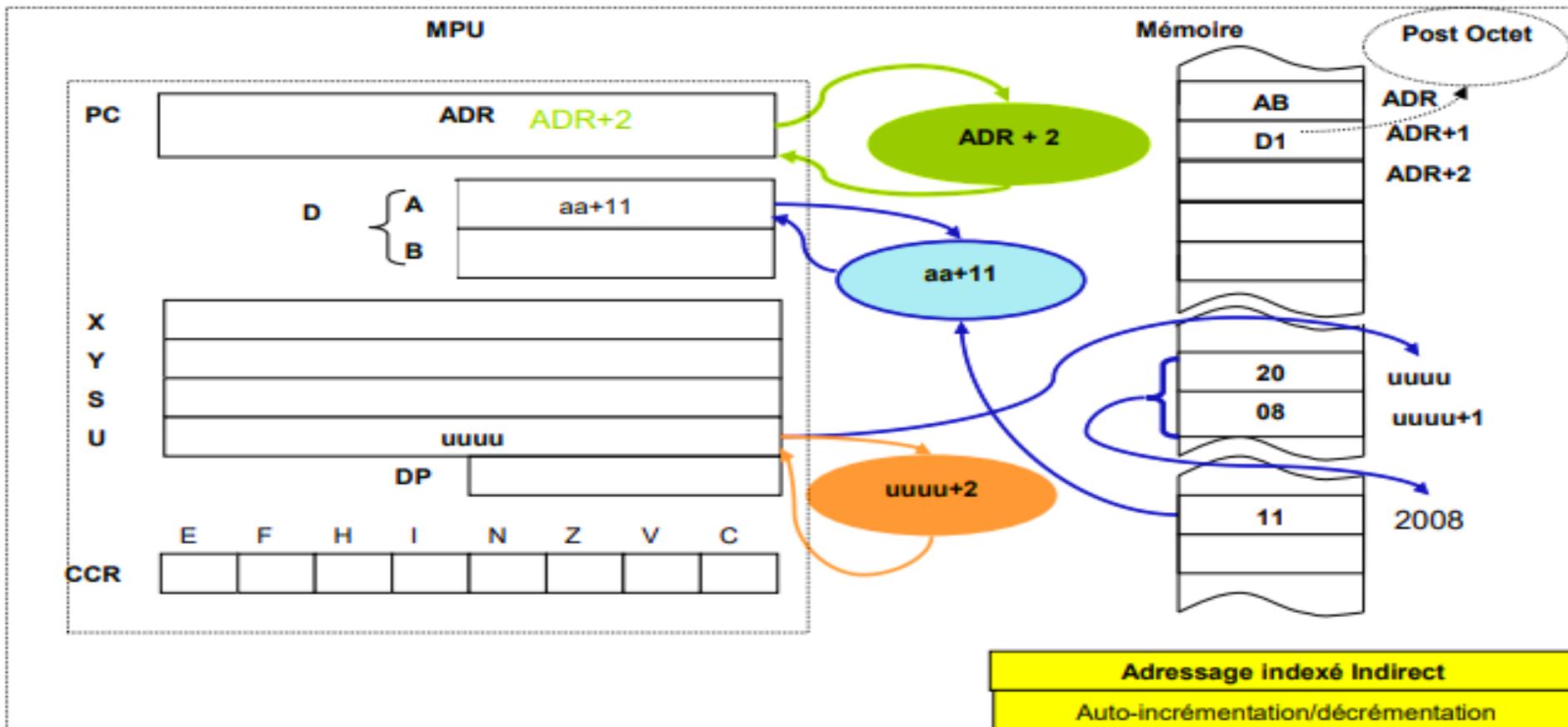
- b7=1 → b4=1 → adressage indexé indirect.
- b6.b5 = 0.0 → index X
- b3.b2.b1.b0=0.1.0.0 → déplacement nul

1	0	0	1	0	1	0	0
X				Déplacement nul			

Adressage indexé indirect

Mode d'adressage indexé indirect. Auto-incrémentation/décrémentation

ADDA [, U++] → addition de l'accumulateur A et du contenu mémoire dont l'adresse se trouve pointée par U,U+1, le résultat est dans A.



Le post-octet prend la valeur **D1** en hexa (1101 0001) :

- b7=1 → b4=1 → adressage indexé indirect.
- b6.b5 =1.0 → index U
- b3.b2.b1.b0=0.0.0.1 → post auto_incrémentation de 2



Adressage indexé indirect

Mode d'adressage indexé indirect

Mode adressage indexé indirect	1. Déplacement nul	LDA	[,X]
	2. Auto Incrémentation/décrémentation	ADDB	[,U++]
	3. Déplacement constant	LDA	[\$2000,X]
	4 .Déplacement accumulateur	LDU	[D,PC]
	5. Base=Compteur Programme PC	LDA	[\$F000,PC] LDA [TABLE,PC]

CLASSIFICATION DES INSTRUCTIONS

Classification des instructions

Les instructions du 6809 sont classées par groupe (5) :

- I. Instructions de traitement des données
- II. Instructions de transfert
- III. Instructions de test et de branchements
- IV. Instruction opérant sur les pointeurs
- V. Traitement des interruptions

Classification des instructions

I. Instructions de traitement des données

Les instructions de traitement des données se répartissent en Quatre catégories:

1. *Les instructions arithmétiques*
2. *Les instructions de rotation et décalage*
3. *Les instructions logiques*
4. *Les instructions d'incrémentation-décrémentation, mise à 0, complémentation*

Classification des instructions

1. Les instructions arithmétiques

Instruction	Fonction
ADD	Addition du contenu mémoire à un accumulateur
ADC	Addition du contenu mémoire à un accumulateur avec retenue
ABX	Addition de l'accumulateur B à X (Non signé)
DAA	Ajustement décimal de l'accumulateur A
MUL	Multiplication de A par B (Non signé)
SUB	Soustraction du contenu mémoire à l'accumulateur
SBC	Soustraction du contenu mémoire à l'accumulateur avec retenue

Classification des instructions

2. Les instructions de rotation et décalage

Instruction	Fonction
ASR	Décalage arithmétique à droite
LSL ou ASL	Décalage logique ou arithmétique à gauche
LSR	Décalage logique à droite
ROL	Rotation à gauche
ROR	Rotation à droite

N.B: Ces instructions opèrent sur des déplacements de 1 bit seulement

Classification des instructions

3 .les instructions logiques

Ces instructions portent toutes sur des données de 8 bits

Instruction	Fonction
AND	« ET logique » entre mémoire et registre interne
EOR	« OU Exclusif » entre mémoire et registre interne
OR	« OU logique» entre mémoire et registre interne

Classification des instructions

4 Instructions d'incrémentation-décrémentation, mise à 0, complémentation

Ces instructions (Sauf NOP) agissent sur une seule opérande.

Instruction	Fonction
CLR	Remise à zéro du contenu mémoire ou de l'accumulateur
DEC	Décrémentation du contenu mémoire ou de l'accumulateur
INC	Incrémentation du contenu mémoire ou de l'accumulateur
NOP	Pas d'opération. Incrémentation du compteur programme
COM	Complément à un du contenu mémoire ou de l'accumulateur
NEG	Complément à deux du contenu mémoire ou de l'accumulateur

Classification des instructions

II Instructions de transfert des données

On peut classer les instructions de transfert des données en TROIS catégories :

1. *Instructions opérant sur les registres internes et la mémoire*
2. *Instructions de transfert opérant sur les pointeurs*
3. *Instructions opérant seulement sur les registres internes du microprocesseur*

1. Instructions opérant sur les registres internes et la mémoire

Ce type d'instructions est très employé. Les transferts mémoire, registre interne du microprocesseur se font sur 8 (A, B) ou sur 16 bits(D, X, Y, S , U). Seuls les registres PC et DP ne sont pas accessibles directement.

Instruction	Fonction
LD	Chargement des registres internes du MPU
ST	Mise en mémoire des registres internes du MPU

Classification des instructions

II Instructions de transfert des données

On peut classer les instructions de transfert des données en TROIS catégories :

1. *Instructions opérant sur les registres internes et la mémoire*
2. *Instructions de transfert opérant sur les pointeurs*
3. *Instructions opérant seulement sur les registres internes du microprocesseur*

2. Instructions de transfert opérant sur les pointeurs

Pour accéder aux piles du 6809, il existe deux instructions « d'empilage » et de « dépileage ». Ces instructions transfèrent n'importe quel registre interne du microprocesseur.

Instruction	Fonction
PSH	Empilement de(s) registre(s) sur la pile
PUL	Dépilement de(s) registre(s)

Classification des instructions

II Instructions de transfert des données

On peut classer les instructions de transfert des données en TROIS catégories :

1. *Instructions opérant sur les registres internes et la mémoire*
2. *Instructions de transfert opérant sur les pointeurs*
3. *Instructions opérant seulement sur les registres internes du microprocesseur*

3. Instructions opérant seulement sur les registres internes du microprocesseur

Ces instructions permettent tous les transferts possibles entre les registres internes du microprocesseur.

Instruction	Fonction
EXG	Echange du contenu de deux registres
TRF	Transfert de registre à registre

Classification des instructions

III Instructions de tests et branchements

Ces instructions peuvent se répartir en trois catégories distinctes :

1. Instructions de test et de comparaison
 2. Instructions de test et de branchement ;
 3. Instructions de saut et branchement
1. Instructions de test et de comparaison

Ces instructions sont utilisées pour réaliser des tests de bits et des comparaisons. Les contenus mémoire et accumulateur ne sont pas modifiés. Le résultat de ces instructions agit uniquement sur les indicateurs du registre d'état, il n'entraîne pas de rupture de séquence(aucun branchement n'est effectué).

Instruction	Fonction
BIT	Test de bits entre accumulateur et contenu mémoire
CMP	Comparaison du contenu mémoire (1 ou 2 octets) avec un registre interne (8 ou 16 bits) du microprocesseur
TST	Test du contenu mémoire ou d'un accumulateur

Classification des instructions

2. Instructions de test et de branchement

Ces instructions permettent de réaliser des branchements conditionnels. Les tests s'effectuent sur 4 indicateurs du registre d'état :

- **N** : Bit de signe ou bit « négatif » (bit 3 du **CCR**) ;
- **Z** : indique si le résultat des opérations est égal ou différent de zéro (bit 2 du **CCR**) ;
- **V** : indicateur de débordement (bit 1 du **CCR**) ;
- **C** : bit de retenue (bit 0 du **CCR**)

Instruction	Fonction
(L)BCC ou (L)BHS	Branchemet si pas de retenue
(L)BCS ou (L)BLO	Branchemet si retenue
(L) BEQ	Branchemet si égale à zéro
(L)BNE	Branchemet si différent de zéro
(L)BGE	Branchemet si supérieur ou égal à zéro
(L)BLT	Branchemet si inférieur (signé)
(L)BGT	Branchemet si supérieur (signé)
(L)BLE	Branchemet si inférieur ou égal (signé)
(L)BHI	Branchemet si supérieur (non signé)
(L)BLS	Branchemet si inférieur ou égal (non signé)
(L)BMI	Branchemet si négatif
(L)BPL	Branchemet si positif
(L)BVC	Branchemet si pas de débordement
(L)BVS	Branchemet si débordement

N.B : La lettre (L) précédant le mnémonique précise qu'il s'agit d'un déplacement long (16 bits).

Classification des instructions

III Instructions de tests et branchements

3. Instructions de saut et branchement

Ces instructions entraînent des ruptures de séquence, sans conditions préalables. Il faut toutefois différencier les instructions de branchement qui sont suivies d'un déplacement long ou court, des instructions de saut qui sont, elles, suivies de l'adresse effective.

Instruction	Fonction
(L)BRA	Branchement inconditionnel
(L)BRN	Non branchement (non opération)
(L) BSR	Branchement à un sous programme
JMP	Saut inconditionnel à une adresse effective
JSR	Saut à un sous programme
RTS	Retour de sous programme

Classification des instructions

IV Instructions opérant sur les pointeurs

Le processeur calcule une adresse effective en fonction du mode d'adressage spécifique(toujours indexé) puis charge cette valeur dans le pointeur (X, Y, S ou U).

Instruction	Fonction
LEA	Chargement d'un registre pointeur avec une adresse effective

V. Traitement des interruptions

Il existe sur le 6809, trois interruptions logicielles et trois interruptions matérielles.

Instruction	Fonction
CWAY	Validation puis attente d'une interruption
SYNC	Synchronisation du logiciel avec une ligne d'interruption
RTI	Retour du sous programme d'interruption
SW1/SW2/SW3	Interruption logicielle